Course 16

Video-based Rendering

Organizers
Marcus Magnor
MPI Informatik

Marc Pollefeys
University of North Carolina at Chapel Hill

Lecturers
German Cheung
Neven Vision Inc.

Wojciech Matusik
Mitsubishi Electric Research Laboratory (MERL)

Christian Theobalt
MPI Informatik

August 1, 2005
ACM Siggraph 2005
Los Angeles, CA USA

Table of Contents

Course Introduction

Summary	III
Syllabus	III
About the course	IV
Prerequisites	IV
Additional resources	IV
Course presenter information	V

Course Slides

Introduction	2
Acquisition	10
Cameras, setup	11
Pre-processing	18
Small-baseline VBR	21
3D TV	22
Dynamic Light Field Warping	63
Wide-baseline VBR	86
Visual Hull Rendering	87
Photo Hull Rendering	133
Camera Calibration	140
Geometric calibration	141
Multi-camera network	147
Unsynchronized video	164
Spacetime Coherence	168
Spatiotemporal View Interpolation	169
Spacetime-coherent Reconstruction	178
Model-based VBR	188
Kinematics Modeling	189
Free-Viewpoint Video	201
References	221
Online VBR Resources	228
Presenters' Contact Information	229

Course Summary

This course intends to give a thorough introduction to how to acquire and process multiple video streams for omni-perspective, interactive rendering of real-world, dynamic scenes. Attendees learn how to reconstruct and represent dynamic scene geometry from multi-video footage, as well as how to render time-varying scenes video-realistically from arbitrary viewpoints in real time.

Syllabus

Basic part

8:30 – 8:45	15 min	**Introduction** - *Magnor* What is video-based rendering ? Why bother ?
8:45 – 9:05	20 min	**Acquisition** – *Cheung* Camera hardware Infrastructure Pre-processing
9:05 – 9:45	40 min	**Small-baseline VBR** – *Matusik, Magnor* 3D TV Dynamic light field warping
9:45 – 10:15	30 min	**Wide-baseline VBR** – *Matusik, Pollefeys* Visual hull rendering Photo hull on GPU

Advanced part

10:30 – 11:00	30 min	**Camera calibration** – *Pollefeys* Intrinsic, extrinsic calibration Multi-camera network calibration Unsynchronized multi-video footage
11:00 – 11:30	30 min	**Spacetime coherence** – *Cheung, Magnor* Visual hull across time Scene flow Spacetime-coherent reconstruction
11:30 – 12:00	30 min	**Model-based VBR** – *Cheung, Theobalt* Kinematics modeling Marker-less motion capture Free-Viewpoint Video
12:00 – 12:15	15 min	**Outlook & Discussion** – *Pollefeys, Magnor*

About this course

In recent years, image-based rendering techniques have advanced from static to time-varying scenes. Instead of still images, multiple synchronized video streams now capture the dynamic scene from different viewpoints. Suitably processed, this multi-video footage is all that is needed to seat “couch potatoes” into the movie director's chair: video-based rendering (VBR) techniques let the user interactively navigate his or her viewpoint through the real-world, dynamic scene to experience movie action or sports events from any self-defined perspective.

The course presents the state-of-the-art in video-based rendering research. Our goal is to empower course participants to use VBR techniques for their own projects. During the course, the necessary algorithmic concepts from computer graphics and computer vision are explained. In addition, we provide multi-video sequences and a lot of additional information on our web sites.

The course syllabus is divided into two parts: In the first part, we cover multi-video acquisition and VBR techniques that originate from still image-based rendering approaches. After the break, more advanced topics are explained, including multi-camera calibration, VBR algorithms that take the temporal dimension into account, as well as approaches that make use of a-priori model information about scene content. We explain the how-to as clearly as possible, and we give an account of the advantages and limitations of each VBR technique.

Besides rendering arbitrary views of real-world, dynamic scenes, VBR techniques are also immensely useful for providing the input modality to data-driven modeling approaches. By offering visually authentic models of dynamic natural phenomena, VBR is a key technology to synthesize realistic animations from real-world examples.

Prerequisites

Participants should be familiar with the concept of image-based rendering. Some basic knowledge of computer vision fundamentals is helpful, but not mandatory.

The course is geared towards graduate students interested in interdisciplinary graphics and vision research, as well as professionals from the movie/special effects industry and interactive entertainment industry.

Additional resources

The following web site has been set up to include links to on-line VBR resources and research groups worldwide that are active in VBR:

http://www.video-based-rendering.org

The web site is maintained by the course organizer to be up to date with latest developments in VBR research. New references to not-yet linked web sites are highly welcome !

Course Presenter Information

Kong (German) Cheung received his Master of Philosophy in Electrical and Electronic Engineering from Hong Kong University of Science and Technology in 1994 and his PhD in Robotics from the Robotics Institute of Carnegie Mellon University (CMU) in 2003, majoring in computer vision and graphics. While at CMU, German managed and conducted research in the pioneering Virtualized Reality(Tm) Laboratory, a facility at CMU with over 50 synchronized, networked cameras capable of capturing dynamic events in real-time. His research interests include 3D shape reconstruction, dynamic scene analysis and re-rendering, human kinematic modeling, markerless motion capture,
video-based rendering and computer vision for graphics applications. After completing his Postdoctoral Fellowship in the Robotics Institute in 2004, German joins Neven Vision Inc., a leading face recognition and tracking company in Santa Monica California, as a senior research scientist.

Marcus Magnor is head of the independent research group Graphics-Optics-Vision at the Max-Planck-Institut Informatik in Saarbrucken, Germany. He received his B.S. in physics in 1995 from the University of Wurzburg, Germany, and his M.Sc. in physics from the University of New Mexico, USA, in 1997. He then joined Bernd Girod's Telecommunications Lab at the University of Erlangen, Germany, where he received his Ph.D. in Electrical Engineering in 2000. While working the following year as Research Associate in the Computer Graphics Lab at Stanford University, USA, he got involved in dynamic light field processing. His research interests include video-based rendering, realistic visualization, as well as data-based modeling techniques. He is currently lecturing courses on computer graphics and computer vision at Saarbrucken University, Germany.

Wojciech Matusik is a visiting research scientist at Mitsubishi Electric Research Labs. He received a B.S. in EECS from the University of California at Berkeley in 1997, M.S. in EECS from MIT in 2001, and consequently Ph.D. in 2003. His primary research lies in computer graphics with an emphasis on modeling based on measured data. He also works on image-based rendering, modeling, and lighting where he developed efficient algorithms for computing and rendering visual hulls. He is currently developing end-to-end 3D TV systems.

Marc Pollefeys is an Assistant Professor of Computer Science at the University of North Carolina at Chapel Hill. He received his M.S. and Ph.D. degrees from the K.U.Leuven in 1994 and 1999, respectively. His main area of research is computer vision, more specifically multi-view geometry, structure from motion, (self-)calibration, stereo, image-based rendering and applications. One of his main research goals is to develop flexible approaches to capture visual representations of real world objects, scenes and events. Dr. Pollefeys has received several prizes for his research, including the prestigious Marr prize at ICCV '98. Recently he also obtained an NSF career award. He is the author or co-author of more than 60

peer-reviewed technical papers. He has organized several workshops, has been serving on the program committees of major conferences such as ICCV, CVPR and ECCV and is a regular reviewer for most of the major vision, graphics and photogrammetry journals. He has organized courses on "Obtaining 3D Models With a Hand-Held Camera" at Siggraph 2000, 2001, 2002; and "3D Models from Photos and Videos" at Siggraph 2003. He has also participated in the courses on "Image-based modeling/Acquisition and Visualization of Surface Lightfields" organized at Siggraph 2001/2002 by Radek Grzeszczuk and in the course on "Interactive Geometric and Scientific Computations Using Graphics Hardware" organized by Dinesh Manocha.

Christian Theobalt received his M.Sc. degree in Artificial Intelligence from the University of Edinburgh, Scotland, and his Diplom (M.S.) degree in Computer Science from the Saarland University, Saarbrucken, Germany, in 2000 and 2001, respectively. Christian is a final year PhD student in the Computer Graphics Group at the MPI Informatik, Saarbrucken, Germany. His research interests include image- and video-based motion analysis, 3D computer vision, and image- and video-based rendering and reconstruction.

Video-based Rendering

German Cheung

Marcus Magnor

Wojciech Matusik

Marc Pollefeys

Christian Theobalt

Introduction
8:30 – 8:45

Marcus Magnor

The Ultimate Goal

SIGGRAPH2005

Scene from “The Matrix”, © Warner Bros. 2000

To the general public (and to most of us, too), computer graphics rendering is about creating spectacular visual impressions. As such, one application area of computer graphics is the generation of special effects for the movie industry where absolutely realistic-looking image sequences of imaginary worlds and/or “impossible” fly-throughs of seemingly real scenes must to be created. One would expect that modern technology is the key, offering utmost flexibility to the director and artistic designers.

Behind the Scenes …

http://whatisthematrix.warnerbros.com/

Reality, however, often looks a lot different. Conventional photographic techniques offer much easier ways to create realistically-appearing images, and computer graphics techniques only augment f/x production. A lot of creative post-processing flexibility is sacrificed this way, in this example by restricting possible later viewpoints to camera recording positions during the shot.

Photography vs. Computer Graphics

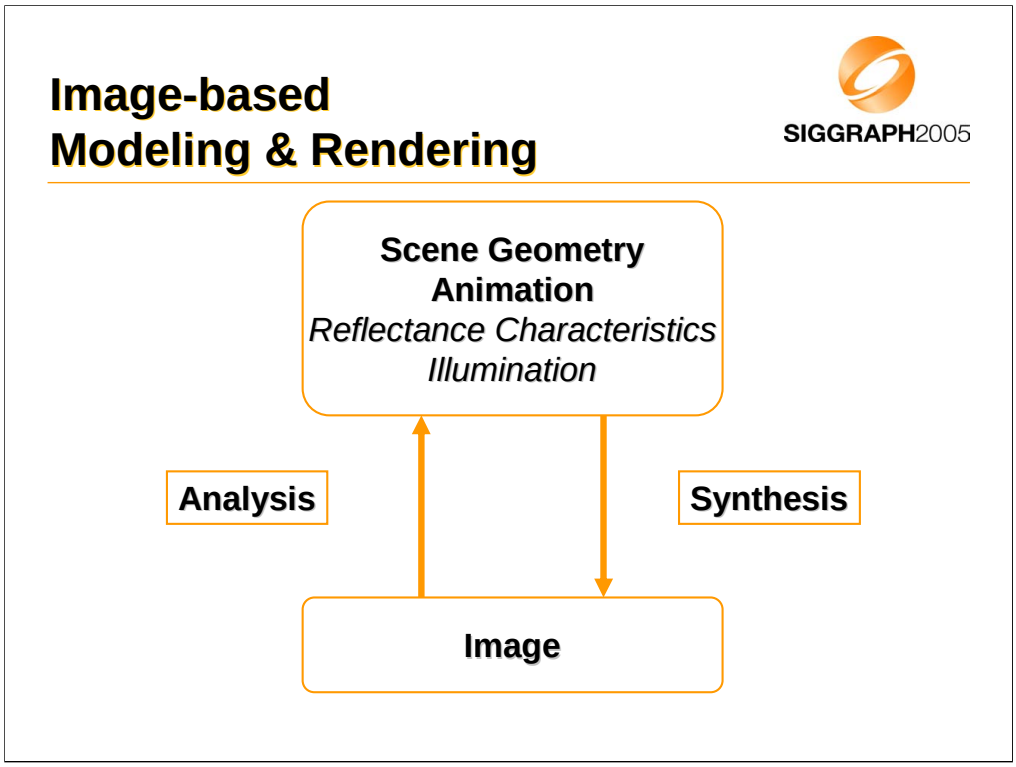
SIGGRAPH2005

+ *Easy acquisition*
+ *Fast display*
+ *Natural appearance*
– *No flexibility*

+ *Unlimited flexibility*
– *Time-consuming modeling*
– *Expensive image synthesis*
– *Non-realistic appearance*

Photographic techniques offer a very fast and inexpensive way to capture the natural visual impression of real-world scenes. Photos can be displayed almost immediately, independent of depicted scene complexity. However, the scene can only be displayed as recorded. Altering the scene or the viewpoint are impossible.

Computer graphics rendering, on the other hand, allows manipulating the scene as well as the viewpoint almost arbitrarily. The price you have to pay for this flexibility, however, is the considerable effort needed to model the scene in terms of its abstract constituents: geometry, surface reflectance characteristics, illumination, and possibly animation. In addition, image synthesis consists of calculating the light distribution in the scene, possibly a very complex, time-consuming task (=> global illumination). Finally, despite a lot of effort, computer graphics-generated images may still have a somewhat artificial flavor if a scene would naturally display subtle visual cues that are missing from the rendered image.



We want to combine the best of both worlds, photography and computer graphics. That is, we want to exploit the ease and naturalness of photographic acquisition and combine it with at least some of the flexibility that computer graphics rendering can offer.

Over the last decade, various image-based modeling and rendering techniques have been developed towards this goal. Based on conventional photos, a description of scene content in terms of geometry is derived. Some approaches even estimate reflectance and scene illumination. From this more abstract scene description, rendering techniques are able to create novel views of the recorded scene.

Recovering model information from image data is traditionally part of computer vision research. With the development of image-based rendering techniques, the reconstruction of visually plausible models from image data has also become an important, active research area in graphics. Researchers from computer graphics and computer vision are working together to recover scene geometry, animation data (via non-invasive optical motion capture), surface characteristics and illumination conditions (inverse rendering) from images and video footage.

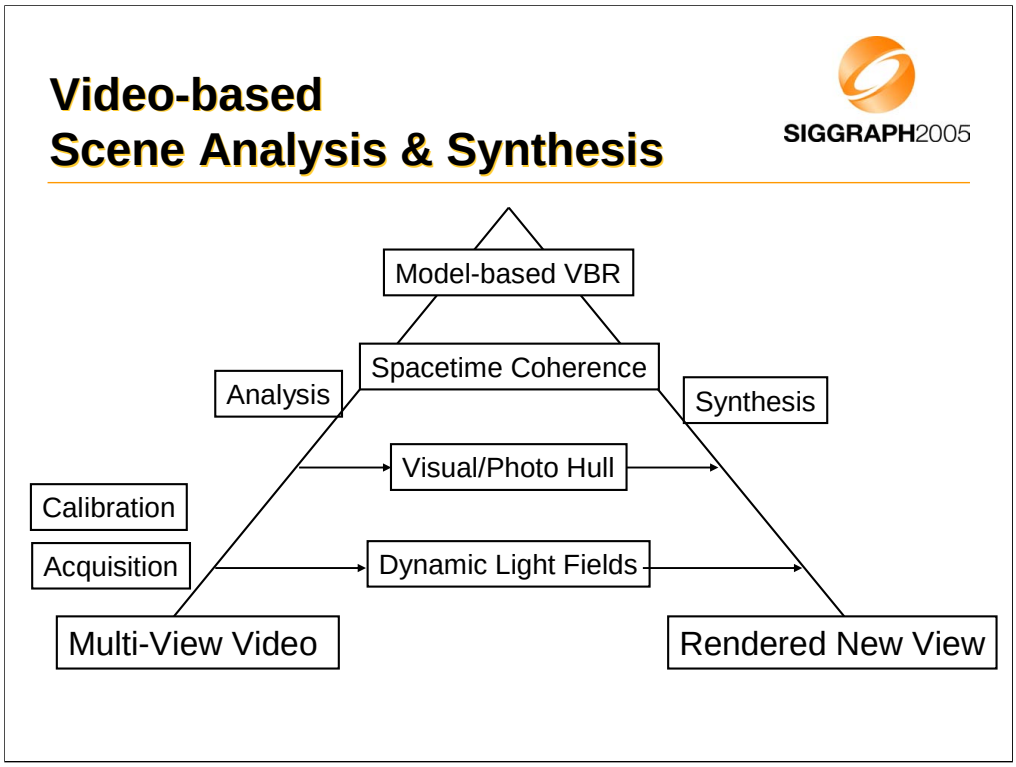
In this tutorial, we limit ourselves to describe how to recover time-varying geometry of a dynamic scene from synchronized video recordings. By using the recorded video images as texture, the scene can be re-displayed from arbitrary viewpoint. Scene illumination, however, remains fixed to the conditions present during recording. Ways to additionally recover surface reflectance and possibly also scene illumination is beyond the scope of our tutorial.

Video-based Rendering

- **Goal**
  - Render dynamic, real-world scene from arbitrary viewpoint
- **Means**
  - Reconstruct dynamic 3D shape from multiple video streams
  - Fast, realistic rendering from shape + video
- **Scenario**
  - Acquisition: synchronized video cameras
  - Processing: consumer-market PC(s)
  - Display: consumer-market PC, graphics hardware
- **Applications**
  - Visual media (“Interactive Television”, “3D TV”)
  - AR/VR (import realism into virtual worlds)

In the following, we will look at different approaches how to render real-world, dynamic scenes at real-time frame rates from novel viewpoints. All discussed VBR techniques rely solely on calibrated, synchronized multi-video footage as input, and all algorithms perform on consumer-market hardware. However, there is no “one size fits all” algorithm in VBR (at least, not yet). Specific algorithms are needed for different application scenarios, e.g., with respect to camera setup (small baseline vs. wide baseline) and processing performance (on-line vs. off-line).

The most immediate applications of VBR are in visual entertainment. For example, VBR may facilitate the TV set of the future to feature a joystick with which the viewer can freely navigate his/her viewpoint all around and through the scene. But also for VR applications, VBR offers importing genuine visual realism of dynamic objects into virtual environments, rendering training simulators, computer games, etc. even more authentic.



VBR starts with multi-video acquisition. Because attainable rendering quality is determined by the quality of the input data, a number of issues must be considered during the recording process.

To relate image data from different video cameras for processing, camera positions, orientations, and imaging characteristics must be accurately determined on-site.

Changing the viewpoint on a dynamic scene can be achieved in different ways. If many cameras are positioned sufficiently close together, the viewpoint can be changed by suitably re-sampling the dynamic light field (3D TV). Large numbers of cameras are needed for meaningful angular coverage. If recording cameras are spaced farther apart, parallax/disparity becomes obvious between adjacent camera views. By estimating dense depth maps per image, disparity can be compensated for during rendering (Dynamic Light Field Warping). Still, the number of cameras necessary to estimate dense depth robustly and to cover large viewing angles isconsiderable.

To reconstruct scene shape from only a handful of video recordings, complete 3D geometry must be considered. Approximate shape can be recovered robustly and fast (visual hull). For more accurate geometry, photo-consistency must be made use of (photo hull).

Specifically in video-based rendering, temporal coherence can be exploited to achieve robust and consistent photo-consistent reconstructions (spacetime coherence).

Finally, if scene content is known a-priori, a parameterized geometry model may be matched to multi-video footage to obtain a high-quality shape description (model-based VBR).

Course Outline

- 8:45 – 9:05 Acquisition
- 9:05 – 9:45 Small-baseline VBR
- 9:45 – 10:15 Large-baseline VBR
- 10:15 – 10:30 Break
- 10:30 – 11:00 Camera Calibration
- 11:00 – 11:30 Spacetime Coherence
- 11:30 – 12:00 Model-based VBR
- 12:00 – 12:15 Summary & Outlook

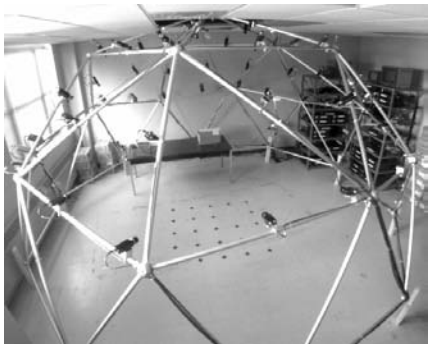
Acquisition
8:45 – 9:05

German Cheung

Acquisition

SIGGRAPH2005

- Very important!!
  - Good acquisition system => good videos => less artifacts + simpler processing algorithms + better final results
  - Cameras
  - Infra-structure
  - The capture
  - Post-capture processing



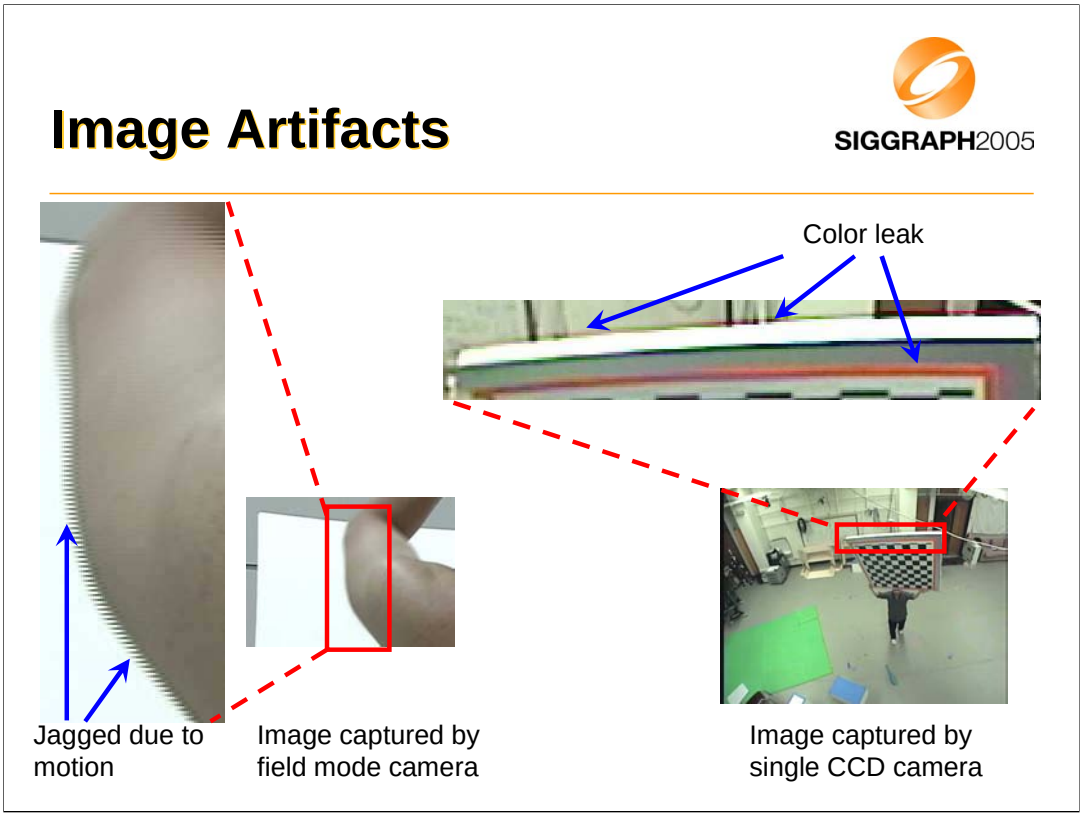
Images courtesy of CMU Virtualized Reality™ Lab.

A good acquisition system is as important as good algorithms and software for image and video-based rendering. High quality video images often eliminate the need of complicated processing algorithms and artist/manual intervention and generally produce better results with less artifacts. Building a good acquisition system involves choosing the right camera(s) and designing the capture infrastructure. The actual capture and post-capture processing are equally important.

Choosing The Right Camera

- Major properties to consider:
  - Digital vs. analogue
  - Resolution vs. bandwidth & processing time
  - Frame mode vs. field mode (jagged images)
  - 3 CCD vs. 1 CCD (color leaking especially bad for blue screening)
  - Easy color adjustment for color calibration

There are several major considerations when choosing the right camera for your applications. Analogue cameras generally capture better images than digital cameras although the latter are becoming more and more popular because they are simpler to set up (without the need of extra capture cards). The higher the resolution of the video, the more details of the scene is captured but it also means higher bandwidth both during capture and processing. Frame mode (non-interlaced) cameras are highly preferred over field mode (interlaced) cameras because the latter ones produce jagged images when capturing fast motion. Moreover, cameras with 1 CCD (as compared to 3 CCD) tend to have color leaking artifacts which are especially bad for background subtraction and blue screening (see next slide for examples of field-mode, color leaking artifacts). The ability to adjust color response is also important for multiple cameras system because cameras with vastly different color responses will produce mismatching color artifacts, especially when view-dependent rendering algorithms are used.

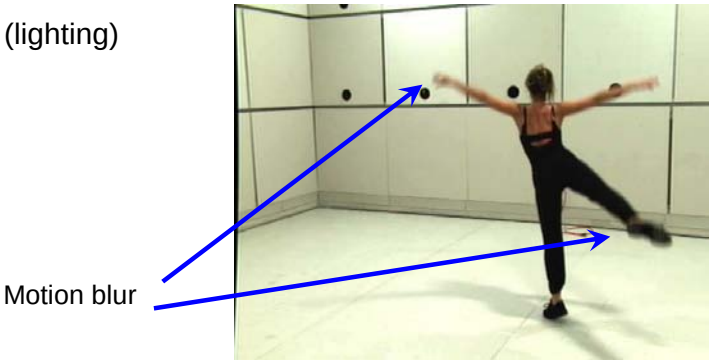


The picture in the left are taken with a camera in field mode. The interlaced odd and even fields create jagged image of the moving arm. The picture in the right is captured by a camera with only one CCD. There is “color leaking” (rings of RGB) from one pixel to another.

Cameras

SIGGRAPH2005

- Other considerations
  - Frame rate (fast motion vs. slow motion)
  - Lens (zoom in/out, wide angle vs lens distortion)
  - Shutter speed vs Motion blur
  - Dynamic range (lighting)



Other minor factors to be considered for camera selection include lens selection (lens distortion effect caused by wide-angle lens), frame rate (frame rate higher than 60Hz is desired when capturing very fast motion such as baseball pitching and golf swinging), shutter speed adjustment to avoid motion blur (see the attached picture) and finally the dynamic range of the camera with respect to extreme lighting of the scene.

The Infra-structure

- Single vs Multiple (how many?) cameras
- Camera placement
  - Wide base-line (visual hull approach): cameras placed as “orthogonal” to each other as possible
  - Narrow base-line (stereo): pair cameras
  - adding overhead cameras always help!
- Bandwidth, time-code, synchronization
- Lighting (as even and diffused as possible)
- Capturing software: multiple cameras/pc communications
- Geometric + color calibration of all cameras

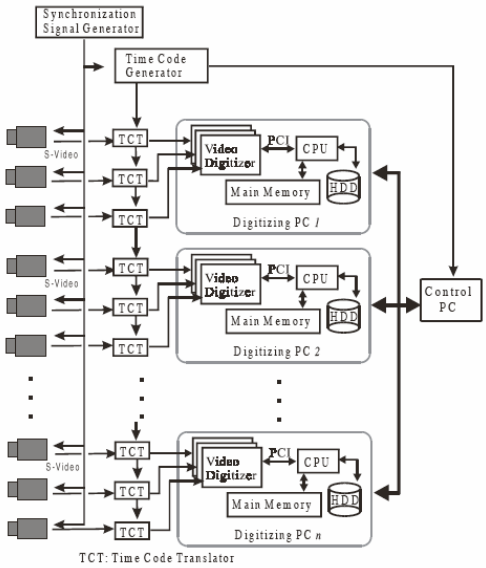
Besides choosing a suitable camera, a good infra-structure is also important. The processing algorithms will probably determine the number of cameras used and their placements (adding overhead cameras almost always help). Camera synchronization and time-code are necessary to make sure the images from multiple cameras are captured and processed correctly time-wise. The bandwidth is always a problem especially for high speed and high resolution cameras (For 10 cameras captured at 30f/s full RGB color with resolution of 640x480 amounts to approximately 270Mbytes/s of data). Lossy compression for faster data transfer and storage (such as jpg) is not recommended especially when blue screening is used. Finally fast and easy geometric calibration and color balancing (which can be difficult to do once there are more than 20 cameras) and reliable communications between the capture PCs are essential requirements of the capture system.

Example System: CMU Virtualized Reality™ Lab.

SIGGRAPH2005

- Started in 1996 as 3D Dome, now 3rd generation
- 24 PCs, 48 synchronized and time-coded cams.
- continuous capture: 640 x 480 resolution, 30f/s for over 2 hrs
- http://www.cs.cmu.edu/~virtualizedr

3D Cage: 3rd generation Virtualized Reality™ Lab.



Images courtesy of CMU Virtualized Reality™ Lab.

The Virtualized Reality™ project was started in 1996 and was considered as the first system of its kind. The image shown in the left is the third generation of the laboratory which consists of 48 cameras controlled by 25 PCs. The cameras are time-coded and synchronized. The system can be used to capture scene at 30f/s at full color and resolution (640x480) continuously for over 2 hours. This facility is an ideal test-bed for performing VBR research. A detailed system design for the second generation of Virtualized Reality™ Lab. can be found in [Kanade et. al. 1998] and its infra-structure is shown in the figure on the right.

[Kanade et. al. 1998] T. Kanade, H. Saito and S. Vedula. The 3D Room: Digitizing time-varying 3D events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, 1998.

The Capture

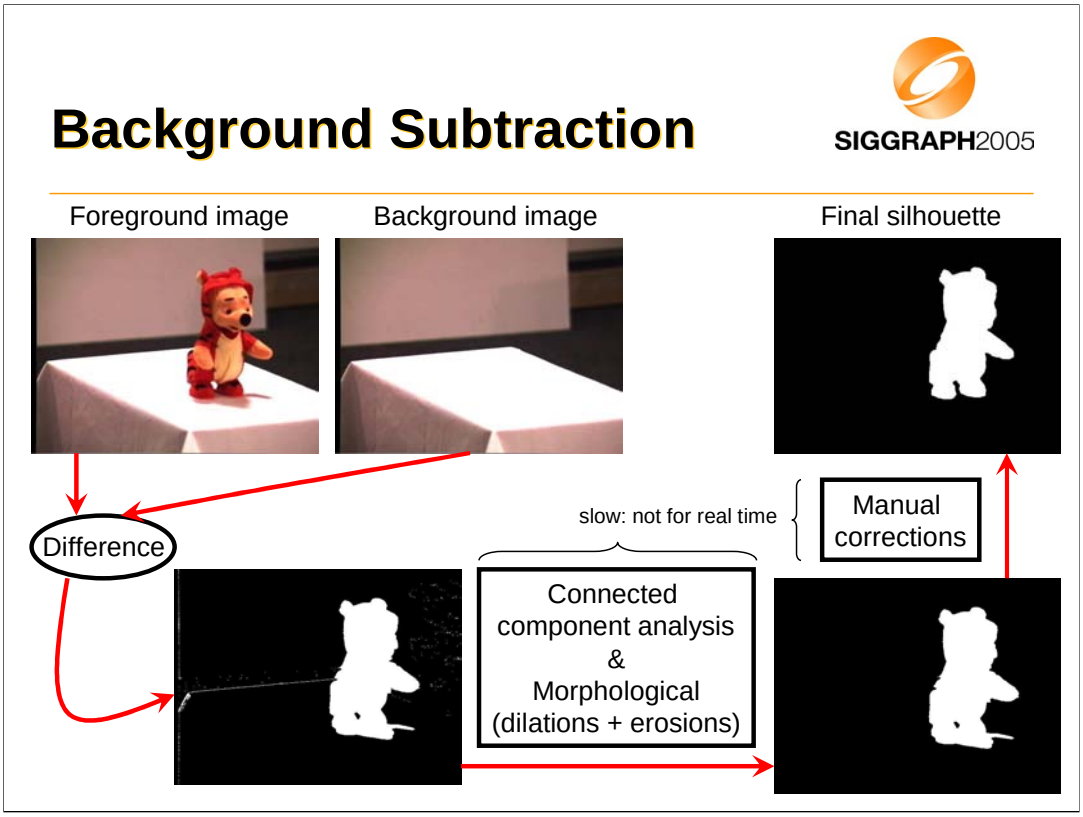
- Plan and rehearse (make sure equipment works especially if you are hiring expensive actors)
- Clean up the background
- Always capture background images (even if you end up not using them)
- Instant playback of all captured videos, if possible
- Capture multiple times

Detailed planning and rehearsal will facilitate smooth operations during the actual capture. While some processing algorithms do not require background images, it is always useful to capture them. Another useful feature during capture is the ability to play back the captured videos (of at least a sub-set of all the cameras) for instant feedback. Finally if possible, capture multiple times and choose the best sequence to process.

Post-capture Processing

- Field mode correction
  - Throw away one field and interpolate the other
- Geometric calibration
  - Recover both extrinsic and intrinsic parameters
- Silhouette generation
  - Construct Visual Hull
  - Restrict search window in stereo
  - Mask out irrelevant information (processing speed)

After the capture, several post-capture steps will help ease subsequent data processing. If field mode cameras are used, it is important to remove the jagged-edge artifacts by removing either the odd or even field and interpolate the other for a full frame image. Geometric calibration is needed before building Visual Hulls (VH) or estimating depth information from the video using stereo. Note that wrong camera calibration can render a whole set of high quality videos useless. Silhouettes are essential in VH construction and useful in reducing for example the search window size in stereo calculation and thus reduce processing time.

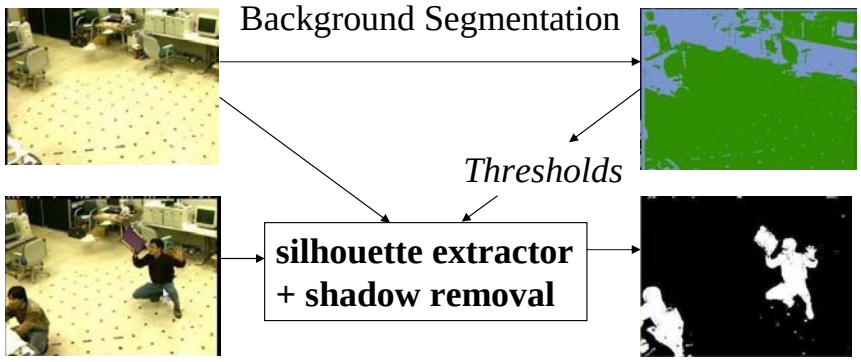


To generate silhouette image from a foreground and a background image, the difference of the two images are compared to a threshold (or a set of thresholds) to classify the image into silhouette and non-silhouette pixels. Connected component analysis (grouping the pixels into connected regions) is very effective in removing spurious pixels wrongly classified in the thresholding process. Morphological operations can then be used to further fill in holes of the silhouette image. Finally for difficult regions such as dark cast shadows, manual corrections may be needed if precise silhouettes are required for subsequent processing. Note that the connected component analysis and morphological operations are slow processes and are not suitable for real-time applications. Chroma-keying such as that in [Smith and Blinn 1996] can also be used if the system is setup with blue screening.

[Smith and Blinn 1996] A. Smith and J. Blinn. Blue Screen Matting. In Computer Graphics Annual Conference Series (Siggraph'96), pages 259-268, 1996.

Region-based Approach

- Fast but robust enough for real-time applications
- Remove shadows on the floor by calculating the angles between the color (RGB) vectors between the background pixel and the run-time pixel



For real-time background subtraction, the region-based approach is quite effective to produce reasonable silhouettes. A pre-processing step is used to collect statistics of the background image with its pixels divided into regions to allow different thresholds for different regions. For shadow removal, the angle between the RGB vectors of the run-time image pixel and the background image pixel is calculated. This angle is usually small for shadowed pixels which are then removed by a color angle threshold. Note that other more sophisticated but complex real-time background subtraction algorithms ([Horprasert et. al. 1999] and [Ivanov et. al. 2000]) can also be used.

[Horprasert et. al. 1999] T. Horprasert, D. Harwood and L. Davis. A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection. In Proceedings of International Conference on Computer Vision (ICCV'99), Frame-rate Workshop, September, 1999.

[Ivanov et. al. 2000] Y. Ivanov, A. Bobick and J. Liu. Fast Lighting Independent Background Subtraction. International Journal on Computer Vision, 37(2):199-207, 2000.

Small-baseline VBR
9:05 - 9:45

Wojciech Matusik

Marcus Magnor

3D TV – Our Vision

- Immersive
- Unobtrusive
- Multi-user

Just like a window!

In 1908, Gabriel Lippmann, who made major contributions to color photography and three-dimensional displays, contemplated producing a display that provides a “window view upon reality” [Lippmann 1908]. Stephen Benton, one of the pioneers of holographic imaging, refined Lippmann’s vision in the 1970s. He set out to design a scalable spatial display system with television-like characteristics, capable of delivering full color, 3D images with proper occlusion relationships. The display should provide images with binocular parallax (i.e., stereoscopic images) that can be viewed from any viewpoint without special glasses. Such displays are called *multiview autostereoscopic* since they naturally provide binocular and motion parallax for multiple observers. *3D video* usually refers to stored animated sequences, whereas *3D TV* includes real-time acquisition, coding, and transmission of dynamic scenes. In this paper we present an end-to-end 3D TV system with 16 independent high-resolution views and autostereoscopic display.

3D TV – Our Goals

- Capture light flowing through a “window”
- Real-time transmission
- Automultiscopic light field display

We present a system for real-time acquisition, transmission, and high-resolution 3D display of dynamic multiview TV content. We use an array of hardware-synchronized cameras to capture multiple perspective views of the scene. We developed a fully distributed architecture with clusters of PCs on the sender and receiver side. We implemented several large, high-resolution 3D displays by using a multi-projector system and lenticular screens with horizontal parallax only. The system is scalable in the number of acquired, transmitted, and displayed video streams. The hardware is relatively inexpensive and consists mostly of commodity components that will further decrease in price. The system architecture is flexible enough to enable a broad range of research in 3D TV. Our system provides enough viewpoints and enough pixels per viewpoint to produce a believable and immersive 3D experience.

Challenges

- Bandwidth
- Processing
- Real-time
- Multiview autostereoscopic display
- End-to-end system
- Automatic setup and calibration

This project comes with many challenges. First, it requires a huge amount of bandwidth and processing power. This is especially true since we would like our system to be real-time. It also requires building a multiview autostereoscopic display. Assembling all the pieces together also proved to be a challenge. Finally, we would like the system to be very easy to set-up and calibrate.

Contributions

- Real-time end-to-end 3D TV system
- Distributed, scalable architecture
- Multiview video rendering
- Computational alignment for 3D displays

We make the following contributions: **System:** the first real-time end-to-end 3D TV system with 16 independent high-resolution views and autostereoscopic display. **Distributed architecture:** In contrast to previous work in multiview video we use a fully distributed architecture for acquisition, compression, transmission, and image display. **Scalability:** The system is completely scalable in the number of acquired, transmitted, and displayed views. **Multiview video rendering:** A new algorithm efficiently renders novel views from multiple dynamic video streams on a cluster of PCs. **High-resolution 3D display:** Our 3D display provides horizontal parallax with 16 independent perspective views at 1024×768 resolution. **Computational alignment for 3D displays:** Image alignment and intensity adjustment of the 3D multiview display are completely automatic using a camera in the loop.

Outline

- Previous Work
- 3D Display
- System Architecture
- Display Calibration
- Rendering

Previous Work – Beginnings

- Stereoscope [Wheatstone 1838]
- Parallax stereogram [Ives 1903]
- Integral display [Lippman 1908]
- Parallax panoramagram [Ives 1928]
- Lenticular sheets, 1930s
- Hologram [Gabor 1948]

A lot of the significant work in this area dates back to 19th and early 20th century. This proves that people have been fascinated with the 3D for quite some time. Parallax displays emit spatially varying directional light. Much of the early 3D display research focused on improvements to Wheatstone's stereoscope. In 1903, F. Ives used a plate with vertical slits as a barrier over an image with alternating strips of left-eye/right-eye images [Ives 1903]. The resulting device is called a *parallax stereogram*. To extend the limited viewing angleand restricted viewing position of stereograms, Kanolt [Kanolt 1918] and H. Ives [Ives 1928] used narrower slits and smaller pitch between the alternating image stripes. These multiview images are called *parallax panoramagrams*. Stereograms and panoramagrams provide only horizontal parallax. In 1908, Lippmann proposed using an array of spherical lenses instead of slits [Lippmann 1908]. This is frequently called a "fly'seye" lens sheet, and the resulting image is called an *integral photograph*. An integral photograph is a true planar lightfield with directionally varying radiance per pixel (lenslet). It is widely acknowledged that the *hologram* was invented by Dennis Gabor in 1948 [Gabor 1948], although the French physicist Aim´e Cotton first described holographic elements in 1901. Holographic techniques were first applied to image display by Leith and Upatnieks in 1962.

Previous Work – Acquisition

- Model-based systems

  [Kanade 97], [Gross 03], [Carranza 03]

- Light-field systems

  [Levoy 96], [Gortler 96], [Schirmacher 01], [Yang 02], [Wilburn 02], [Naemura 02]

- Multiview video compression and transmission

  [Fehn 02], [Magnor 03], [Ramanathan 03], [Yang 02], [Tanimoto 03], [Zitnick 04], [Smolic 03]

One approach to 3D TV is to acquire multiview video from sparsely arranged cameras and to use some model of the scene for view interpolation. A lightfield represents radiance as a function of position and direction in regions of space free of occluders [Levoy and Hanrahan 1996]. Acquisition of dense, dynamic lightfields has only recently become feasible. Multiview video compression has mostly focused on static lightfields (e.g., [Magnor et al. 2003; Ramanathan et al. 2003]). There has been relatively little research on how to compress and transmit multiview video of dynamic scenes in real-time. Most systems compress the multiview video off-line and focus on providing interactive decoding and display. An overview of some early off-line compression approaches can be found in [Javidi and Okano 2002, Chapter 8].

Previous Work – Displays

- Holographic displays

  [St.-Hillaire 95], [Maeno 96], [Kajiki 96], [Stanley 00], [Huebschman 03]

- Volumetric displays

  [McKay 00], [Favalora 01]

- Parallax displays

  [Nakajima 01], [Liao 02], [Moore 96], [Perlin 00]

- Multi-projector displays

  [Raskar 98], [Li 02], [Humphreys 02]

Stephen Benton's Spatial Imaging Group at MIT has been pioneering the development of electronic holography. Their most recent device, the Mark-II Holographic Video Display, uses acoustooptic modulators, beamsplitters, moving mirrors, and lenses to create interactive holograms [St.-Hillaire et al. 1995]. In more recent systems, moving parts have been eliminated by replacing the acousto-optic modulators with LCD [Maeno et al. 1996], focused light arrays [Kajiki et al. 1996], optically-addressed spatial modulators [Stanley et al. 2000], or digital micromirror devices [Huebschman et al. 2003]. Volumetric displays use a medium to fill or scan a three-dimensional space and individually address and illuminate small voxels [McKay et al. 2000; Favalora et al. 2001]. Parallax displays emit spatially varying directional light. Scalable multi-projector display walls have recently become popular [Li et al. 2002; Raskar et al. 1998]. These systems offer very high resolution, flexibility, excellent cost-performance, scalability, and large-format images. Graphics rendering for multi-projector systems can be efficiently parallelized on clusters of PCs using, for example, the Chromium API [Humphreys et al. 2002].

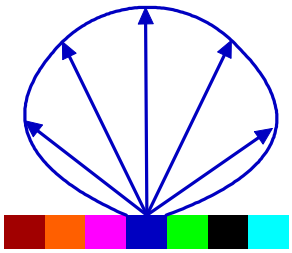
Outline

- Previous Work
- 3D Display
- System Architecture
- Display Calibration
- Rendering

Regular Pixels

SIGGRAPH2005

- Ideally emit the same light in all directions

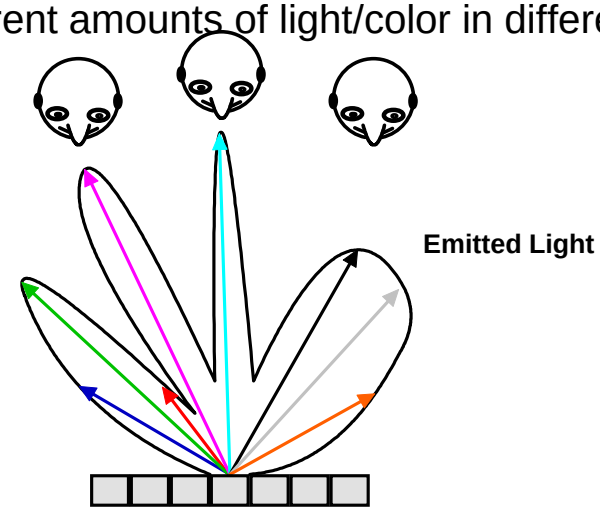


Pixels in regular displays emit the same light in all directions.

View-dependent Pixels

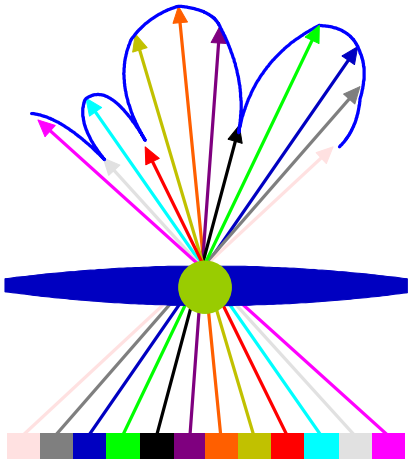
SIGGRAPH2005

- Emit different amounts of light/color in different directions

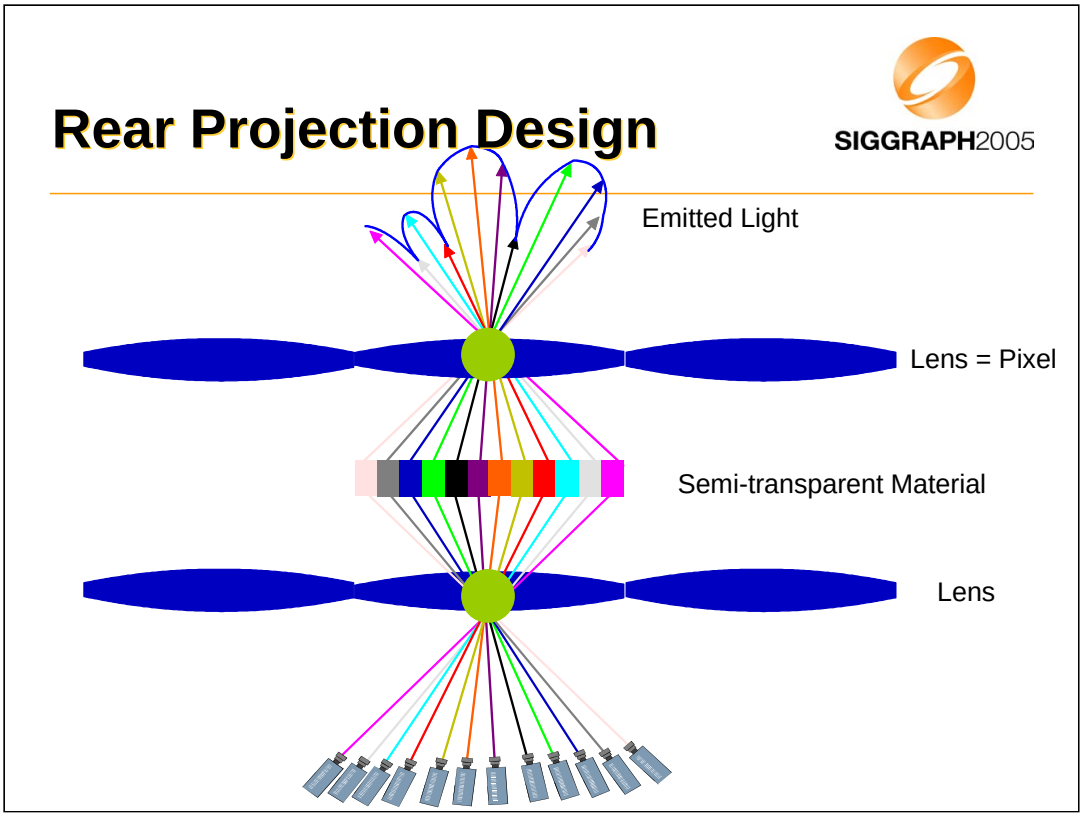


But we would like to have view-dependent pixels. These pixels emit different amount of light in different directions.

View-dependent Pixels

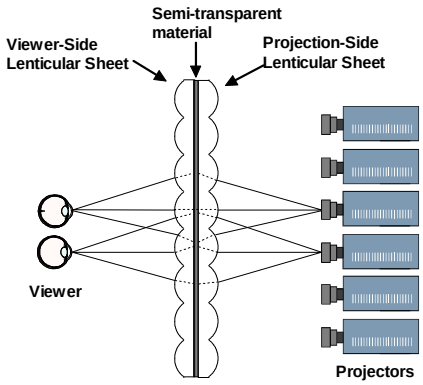


One way to achieve this is to trade the spatial resolution in the display for angular resolution. We put an array of lenses or pinholes on the top of the high resolution screen. This will cause the light from different screen elements to be emitted in different directions.



We have built two types of prototypes. In the rear projection design we have have a semi transparent material sandwiched between to sheets of lenslets. We illuminate the first lenslet sheet from different directions. This creates a high resolution screen on the semi-transparent screen. The light from the high resolution screen on the semi-transparent material is emitted in different directions.

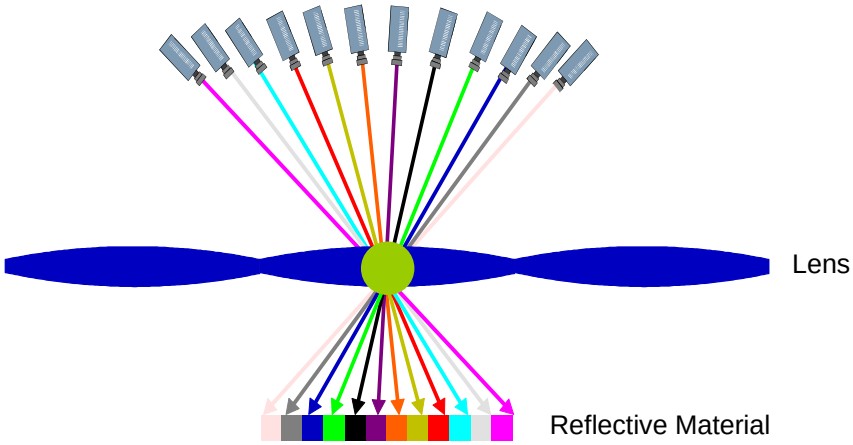
Our Rear Projection Display



This slide show a diagram and a picture of our rear projection display. For the rear-projection system, two lenticular sheets are mounted back-to-back with optical diffuser material in the center. We use a flexible rear-projection fabric from Da-Lite Screen Company (www.da-lite.com). The back-to-back lenticular sheets and the diffuser fabric were composited by Big3D Corp. (www.big3d.com) using transparent resin that was UV-hardened after hand-alignment. The front-projection system uses only one lenticular sheet with a retro-reflective front-projection screen material from Da-Lite mounted on the back.

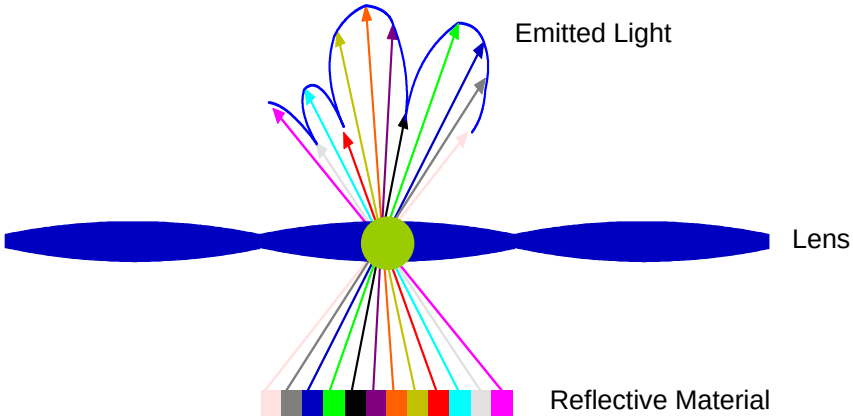
Front Projection Design

SIGGRAPH2005



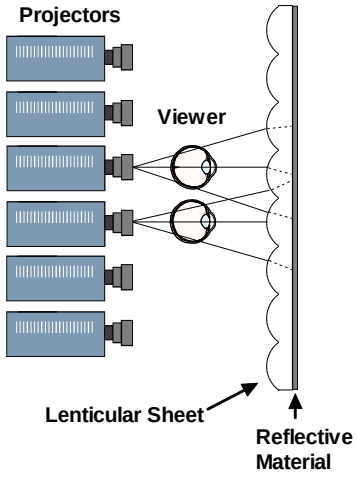
Next we have explored a front-projection design. This is the one we were showing at e-tech last year. We illuminate the first lenslet sheet from different directions. This creates a high resolution screen on the retro-reflective screen.

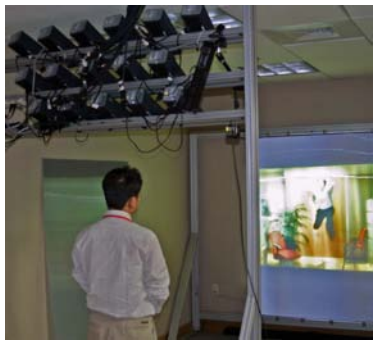
Front Projection Design



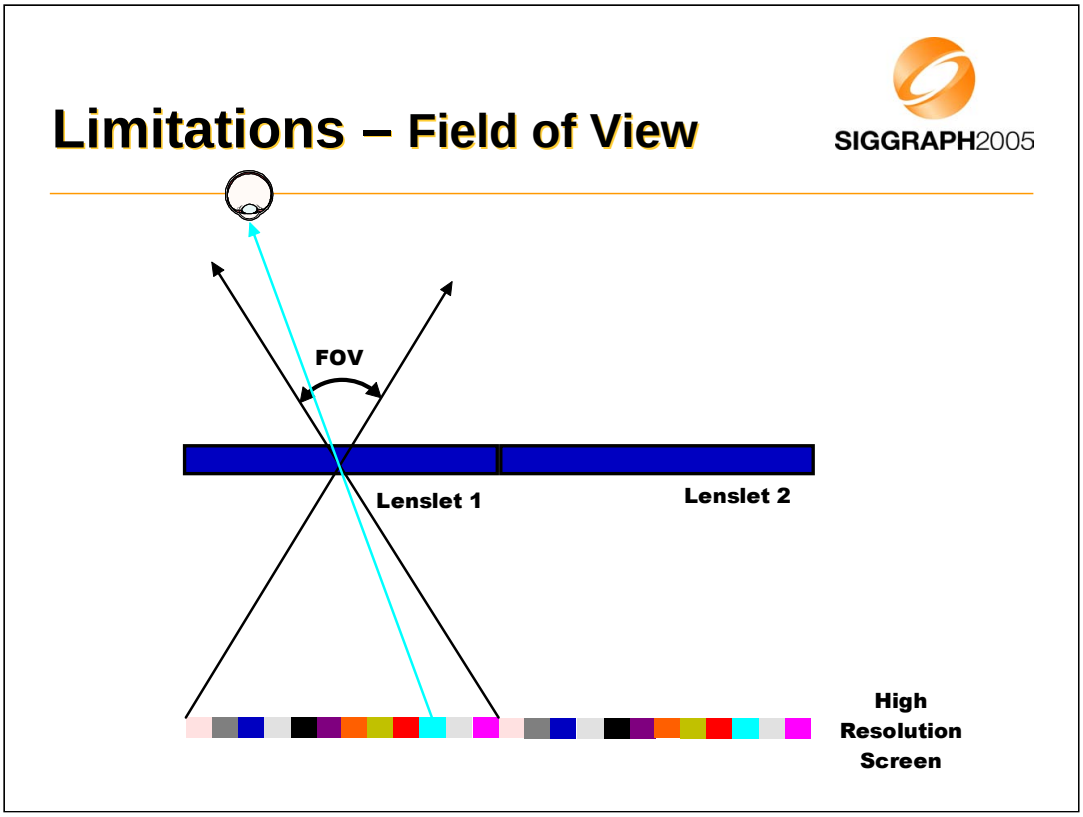
The light is reflected back towards the user.

Our Front Projection Display



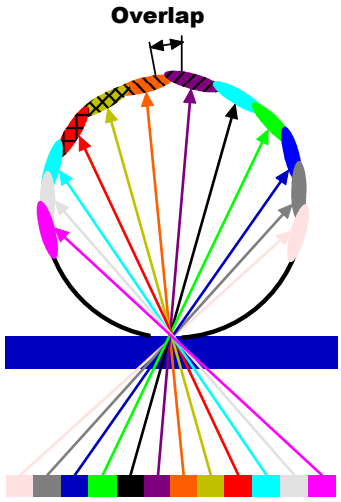


This is the system diagram and the corresponding picture. The projection-side lenticular sheet of the rear-projection display acts as a light multiplexer, focusing the projected light as thin vertical stripes onto the diffuser. The viewer-side lenticular sheet acts as a light de-multiplexer and projects the view-dependent radiance back to the viewer. Note that the single lenticular sheet of the front-projection screen both multiplexes and de-multiplexes the light.



These systems come with some limitations. First of them is the limited field of view. In our case, the FOV is 30 degrees, leading to 180/30 = 6 viewing zones. At the border between two neighboring viewing zones there is an abrupt view-image change (or “jump”) from view number 16 to view number one. The only remedy for this problem is to increase the FOV of the display.

Limitations –
Discretization & Cross-talk



The next limitations are discretization of the angular domain and cross-talk. Angular domain of each view-dependent pixel is discretized into a relatively small number of intervals (in our case 16). This can cause inter-perspective aliasing. Cross-talk occurs when values from a certain view spill over into the neighboring views.

Display Trade-offs

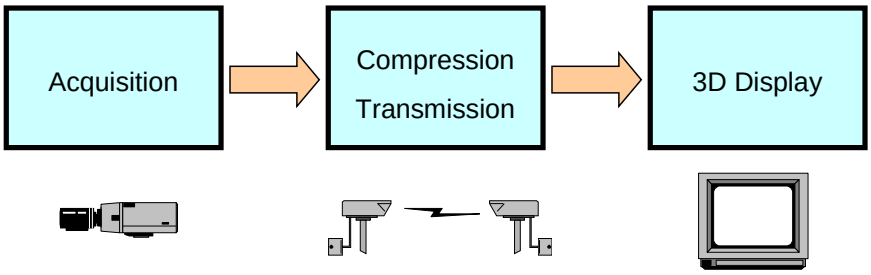
- Horizontal and vertical parallax 3D TV
  - requires $O(n^2)$ bandwidth, computation, & cost
- Horizontal parallax only 3D TV
  - requires O(n) bandwidth, computation, & cost
  - still produces immersive and convincing 3D experience

When designing a 3D display there is a major trade-off. A display that provides both horizontal and vertical parallax requires n*n cameras, bandwidth, computation and cost (where n is the number of views in one direction). A display that provides horizontal parallax requires n cameras, bandwidth, computation, and cost and it still produces and immersive and convincing 3D experience. We have built a display with horizontal only parallax but all system components that we describe next would work for both types.

Outline

- Previous Work
- 3D Display
- System Architecture
- Display Calibration
- Rendering

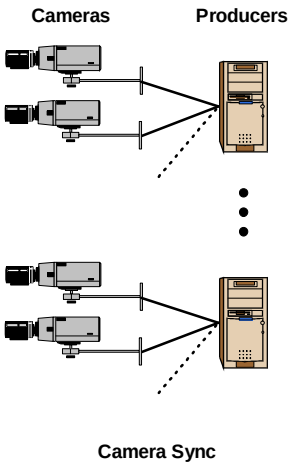
System Architecture



A 3D TV system consists of three major components: (1) Acquisition using a camera array. (2) Compression and Transmission. (3) Display using an automultiscopic display. The *acquisition* stage consists of an array of hardware synchronized cameras. Small clusters of cameras are connected to *producer* PCs. The producers capture live, uncompressed video streams and encode them using standard MPEG coding. The compressed video streams are then broadcast on separate channels over a *transmission* network, which could be digital cable, satellite TV, or the Internet. On the receiver side, individual video streams are decompressed by *decoders*. The decoders are connected by network (e.g., gigabit Ethernet) to a cluster of *consumer* PCs. The consumers render the appropriate views and send them to a standard 2D, stereo-pair 3D, or multiview 3D display.

Acquisition

SIGGRAPH2005



- Array (16) of hardware synchronized, calibrated, cameras
- Distributed acquisition

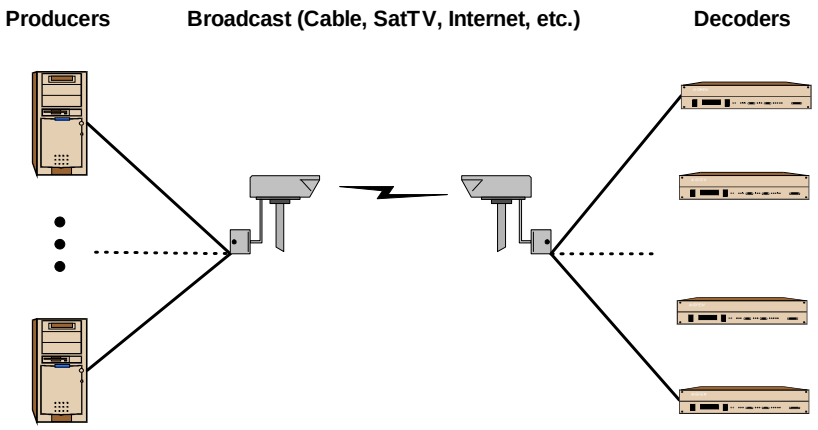
Each camera captures progressive high-definition video in real-time. We are using 16 Basler A101fc color cameras (www.baslerweb.com) with 1300×1030, 8 bits per pixel CCD sensors. The cameras are connected by IEEE-1394 (FireWire) High Performance Serial Bus to the producer PCs. The maximum transmitted frame rate at full resolution is 12 frames per second. Two cameras each are connected to one of eight producer PCs. All PCs in our prototype have 3 GHz Pentium 4 processors, 2 GB of RAM, and run Windows XP. We arranged the 16 cameras in a regularly spaced linear array. The optical axis of each camera is roughly perpendicular to a common camera plane. It is impossible to align multiple cameras precisely, so we use standard calibration procedures [Zhang 2000] to determine the intrinsic and extrinsic camera parameters.

Compression & Transmission

- Temporal encoding
  - Each stream encoded separately
  - Uses existing video standards (MPEG-2)
  - Scalable
- Spatial (multiview) encoding
  - Not scalable
  - Not real-time
- Both temporal & spatial encoding
  - Best compression

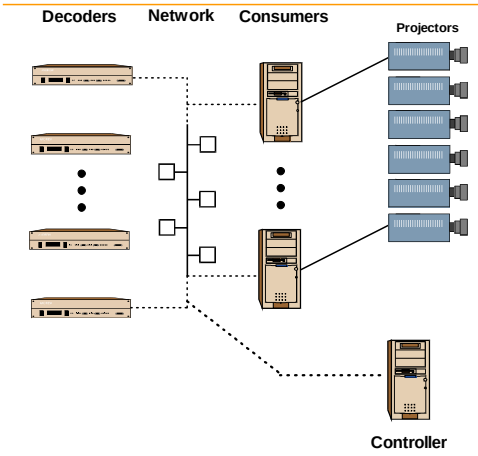
Transmitting 16 uncompressed video streams with 1300×1030 resolution and 24 bits per pixel at 30 frames per second requires 14.4 Gb/sec bandwidth, which is well beyond current broadcast capabilities.For compression and transmission of dynamic multiview video data there are two basic design choices. Either the data from multiple cameras is compressed using spatial or spatio-temporal encoding, or each video stream is compressed individually using temporal encoding. The first option offers higher compression, since there is a lot of coherence between the views. However, it requires that multiple video streams are compressed by a centralized processor. This compression-hub architecture is not scalable, since the addition of more views will eventually overwhelm the internal bandwidth of the encoder. Consequently, we decided to use temporal encoding of individual video streams on distributed processors.

Compression & Transmission



This strategy has other advantages. Existing broadband protocols and compression standards do not need to be changed for immediate real-world 3D TV experiments and market studies. Our system can plug into today's digital TV broadcast infrastructure and co-exist in perfect harmony with 2D TV. Similar to HDTV, the introduction of 3D TV can proceed gradually, with one 3D channel at first and more to follow, depending on market demand. Another advantage of using existing 2D coding standards is that the codecs are well established and widely available. Tomorrow's digital TV set-top box could contain one or many decoders, depending whether the display is 2D or multiview 3D capable. Note that our system can adapt to other 3D TV compression algorithms [Fehn et al. 2002].

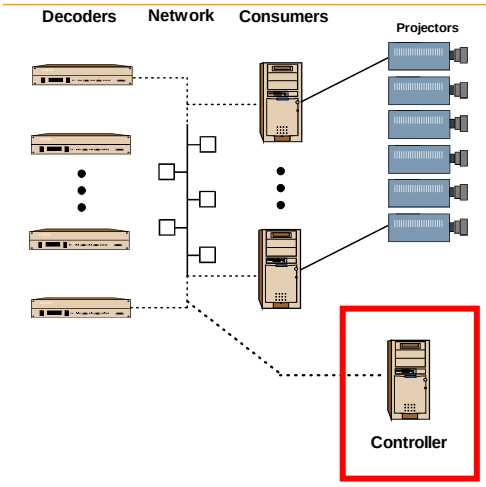
Distributed Display



- Decoders
  - decode video streams
  - send pixel streams to consumers
- Consumers
  - render video streams

The decoders receive a compressed video stream, decode it, and store the current uncompressed source frame in a buffer. Each consumer has a *virtual video buffer (VVB)* with data from *all* current source frames (i.e., all acquired views at a particular time instance). The consumer then generates a complete output image by processing image pixels from multiple frames in the VVB. Due to bandwidth and processing limitations it would be impossible for each consumer to receive the complete source frames from all the decoders. This would also limit the scalability of the system.

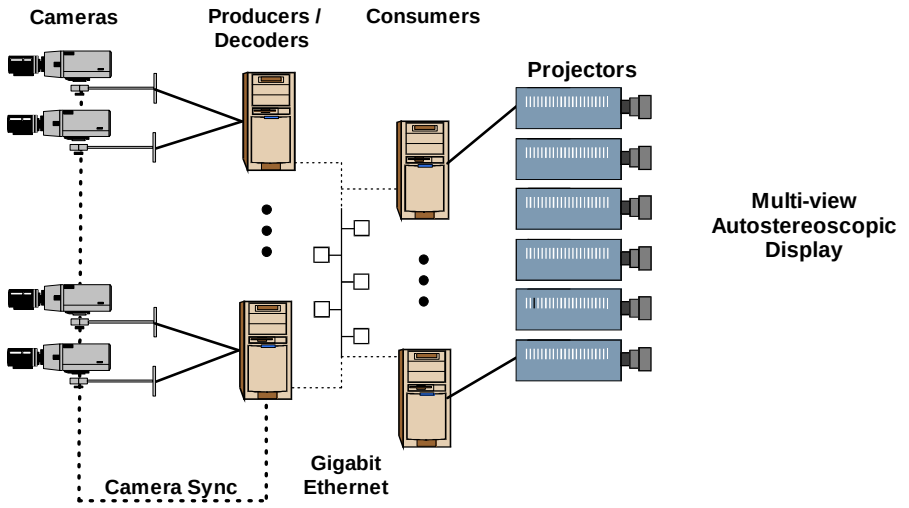
Distributed Display



- Controller
  - decides where to send pixels
  - ensures data flow to each consumer is at most ***k*** *x* video stream (***k*** = 3)
  - allows interactively changing display parameters

A controller PC is a very important component of this subsystem. Its tasks are: deciding where to route pixels & ensuring that data flow to each consumer is at most k* bandwidth of a video stream. It allows us to change display parameters interactively.

Our System



Because we did not have access to digital broadcast equipment, we implemented the modified architecture producer PCs are connected by gigabit Ethernet to eight consumer PCs. Video streams at full camera resolution (1300×1030) are encoded with MPEG-2 and immediately decoded on the producer PCs. This essentially corresponds to a broadband network with infinite bandwidth and almost zero delay. We plan to introduce a more realistic broadband network simulation in the future. The gigabit Ethernet provides all-to-all connectivity between decoders and consumers, which is important for our distributed rendering and display implementation.

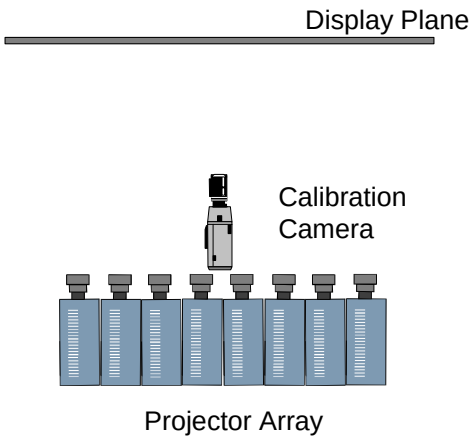
Outline

- Previous Work
- 3D Display
- System Architecture
- Display Calibration
- Rendering

Display Calibration

SIGGRAPH2005

- Geometric calibration
  - Project checkerboard pattern
  - Compute homographies
  - Compute intersection
  - Compute maximum rectangle
- Photometric Calibration

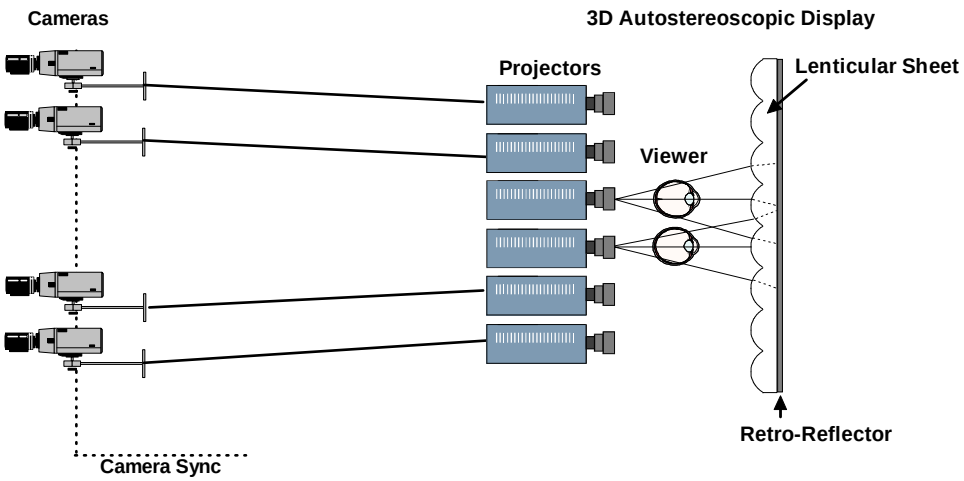


Automatic projector calibration for the 3D display is very important. We first find the relationship between rays in space and pixels in the projected images by placing a camera on the projection side of the screen. Then we equalize the intensities of the projectors. For both processes, the display is covered with a diffuse screen material. We use standard computer vision techniques [Raskar et al. 1999; Li et al. 2002] to find the mapping of points on the display to camera pixels, which (up to unknown scale) can be expressed by a 3× 3 homography matrix. The largest common display area is computed by fitting the largest rectangle of a given aspect ratio (e.g., 4:3) into the intersection of all projected images.

Outline

- Previous Work
- 3D Display
- System Architecture
- Display Calibration
- Rendering

No Rendering



One possible implementation of our system uses a one-to-one mapping of cameras to projectors. In this case, the images need to be rectified using the camera calibration parameters, since the cameras are not accurately aligned. This approach is very simple and scales well.

Simple System Limitations

SIGGRAPH2005

- Physical alignment is difficult
- No flexibility

Unfortunately this does not quite work. This is because the physical alignment of the projectors and cameras is difficult. Furthermore, there is no flexibility at all. For example, suppose that we have a different number of projectors and cameras. You might also need to perform proper filtering to remove anti-aliasing.

Lightfield Rendering

SIGGRAPH2005

- Unstructured Lumigraph Rendering [Buehler 01]
  - Blend 3 rays / pixel
- Closest ray

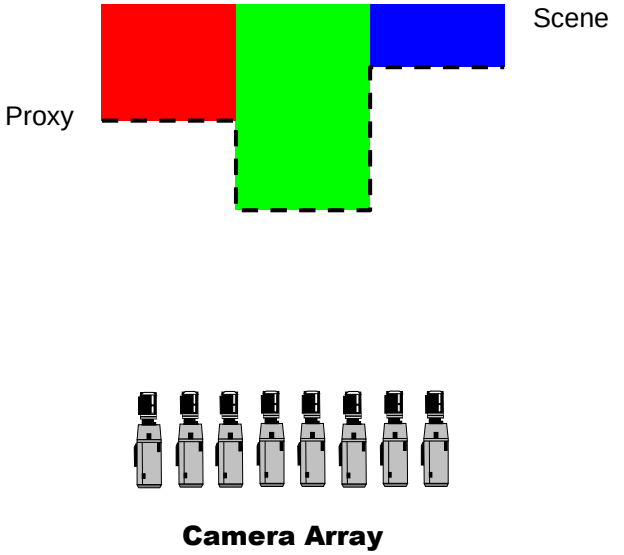
Another, more flexible approach is to use image-based rendering to synthesize views at the correct virtual camera positions. We are using unstructured lumigraph rendering [Buehler et al. 2001] on the consumer side. As in regular lightfield rendering, the geometric proxy for the scene is a single plane that can be set arbitrarily. We choose the plane that is roughly in the center of our depth of field. The virtual viewpoints for the projected images are chosen at even spacings.

Lightfield Rendering

- Flexible and interactive control of
  - Proxy Plane
  - Zero-disparity Plane

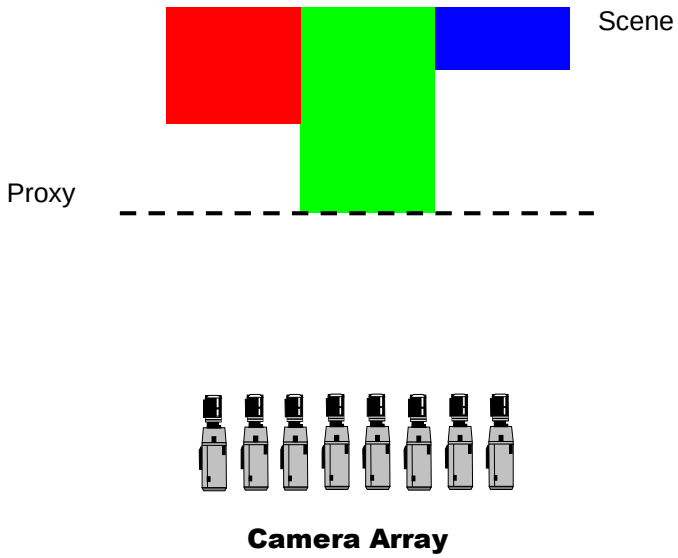
But the main advantage of lightfield rendering is the flexibility how we render the scene. In particular, I will explain how we can change interactively the proxy plane and zero-disparity plane.

Lightfield Rendering



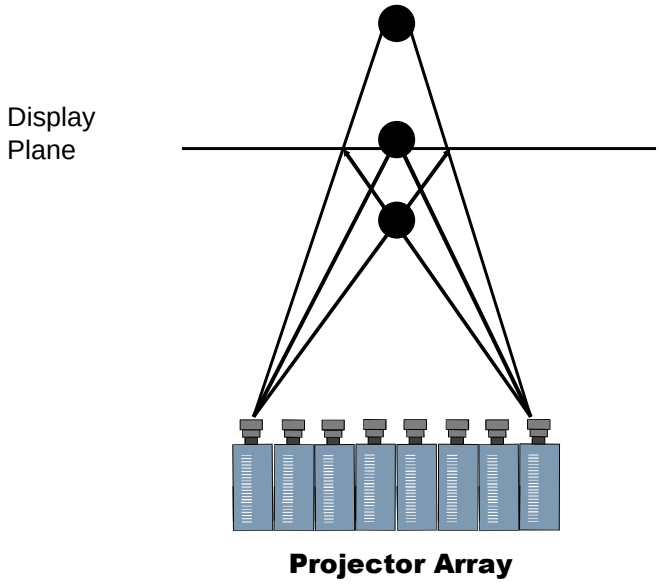
Let's assume we have the following scene. If our geometric proxy for lightfield rendering corresponds to the actual scene geometry. Then we can render views of this scene from arbitrary viewpoints well. In practice, we do not have the actual scene geometry.

Lightfield Rendering



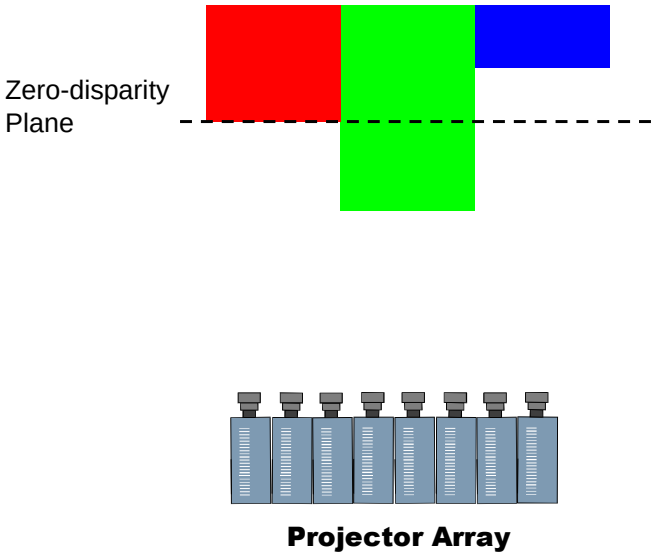
And instead we use a plane as a geometric proxy. Only parts of the scene that are close to this plane will appear sharp. The rest of the scene will be typically blurry. We can interactively change the position of this plane.

Effect of Display Disparities



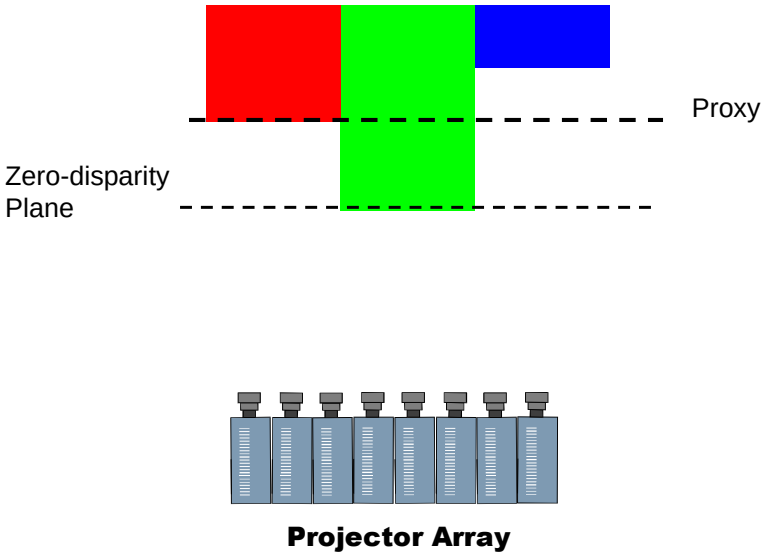
Let's look what happens on the display side. We have a choice what appears in front and what appears behind the physical display plane. A point in front has a disparity which is a distance between displayed pixels. A point behind has a disparity in the other direction as shown in the figure. A point on the display has zero disparity.

Zero-disparity Plane



The points with zero disparity on the display lay on the a virtual plane that goes through the scene. We call this plane the zero disparity plane. We can move this plane backwards and forwards by choosing proper viewpoints in the light field rendering.

Zero-disparity & Proxy Plane



In general the zero disparity plane and the proxy plane do not have to coincide. But it is best if they do so that the parts of the scene that have zero disparity are rendered very sharp and the parts of the scene with more disparity are not as sharp. This method reduces interperspective aliasing.

Conclusions

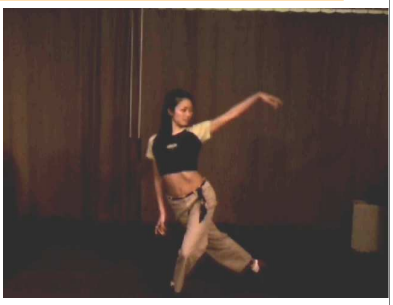
- Real-time end-to-end 3D TV system
- Distributed, scalable architecture
- 3D TV is technically feasible and economically practical today

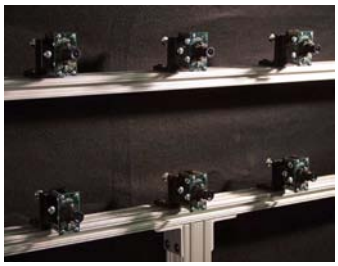
Most of the key ideas for the 3D TV system presented here have been known for decades, such as lenticular screens, multiprojector3D displays, and camera arrays for acquisition. The main advance over previous systems is to a large extent technological; such a system simply could not have been built until cameras and projectors were inexpensive enough and until computation was fast enough. We believe our system is the first to provide enough viewpoints and enough pixels per viewpoint to produce an immersive and convincing 3D experience (at least in the horizontal direction) without special glasses. It is also the first system that provides this experience in real-time for dynamic scenes.

Dynamic Light Field Warping

SIGGRAPH2005

- 2D-array of synchronized video cameras
  - “window" to scene
  - Parallax
- Warping
  - Disparity compensation
  - Dense depth information
- Pre-recorded, static scene background
- Unknown foreground

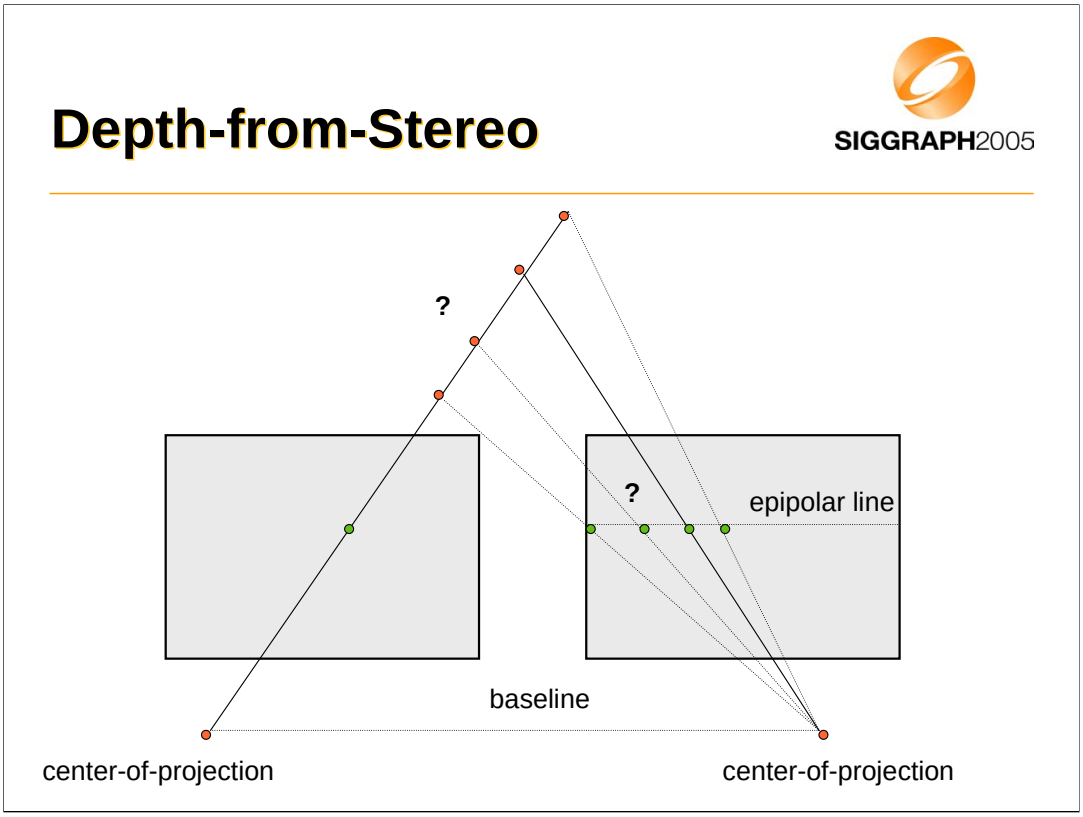




Stanford Multi-Camera Array

In our second example for small-baseline VBR, we now consider the case where our acquisition cameras are still placed close together, but the recorded images already show noticeable disparity, i.e., 3D scene points are projected to 3D image-space coordinates that differ by several pixels. As in the 3D TV system, many video cameras are arranged in a more or less regular grid with all cameras facing in the same direction. Small differences in camera orientation can be correct for using image rectification techniques.

Static scene background is recorded prior to acquiring the unknown, dynamic scene foreground (the dancer in our example). The cameras sample the scene’s dynamic light field. The array of cameras spans a surface we refer to as the recording hull. Dynamic light field warping is able to render the scene from any viewpoint situated on this recording hull.



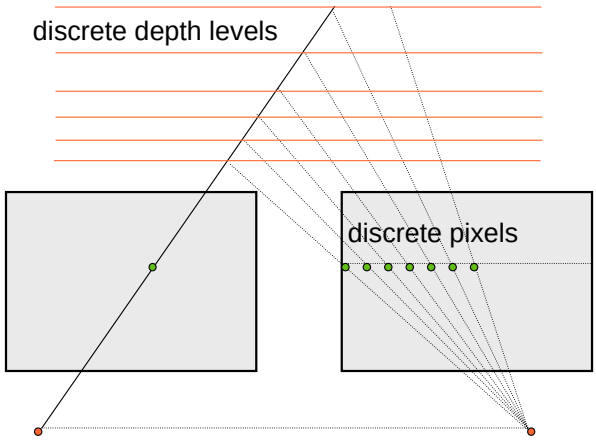
Because adjacent camera images show parallax, we have to be able to compensate for this disparity if we are to render aliasing-free images. For two images whose planes are aligned in parallel, disparity is the difference in pixel coordinates for the same 3D scene point. The amount of disparity is directly proportional to baseline length and inversely proportional to the distance from the camera pair. Through camera calibration, we know the baseline, camera imaging characteristics and therefore also the epipolar geometry.

To compensate disparity for many cameras, it is convenient to first determine the distance of the 3D scene point to the cameras. Per-pixel scene depth is estimated by turning disparity compensation around: for each pixel in the left image, we find the corresponding pixel (along the epipolar line) in the right image. The difference in image coordinates is the point's disparity, and via camera calibration we get its depth. The big challenge in depth-from-stereo is now how to robustly determine image correspondences.

Depth Discretization

SIGGRAPH2005

- Correspondences between image pixels
  - Discrete disparity values
  - Discrete depth levels
- For rectified images
  - minimal distance scene-to-cameras:
    - Finite search range



In the depth estimation approach described in the following, we make use of the fact that for our purpose, scene depth can be discretized into finitely many depth levels. Because we record digital images, pixel correspondences can sensibly be stated only with finite precision. If we determine pixel correspondence by stating only integer pixel positions, the number of pixels along the epipolar line already limits the number of possible depth levels. If the optical axes are aligned in parallel, and the scene is at least a minimum distance away from the cameras, disparity range is further restricted.

In consequence, each pixel is the projection of a 3D scene point that must lie on one of n depth levels.

Multi-View Depth Estimation

- Desirable properties
  - Dense depth maps
  - Robust
  - Smooth
  - Discontinuities preserved
  - Globally consistent
  - All images considered equally
  - Per-pixel foreground/background segmentation

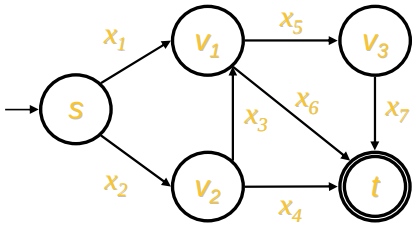
⇒ Combinatorial Optimization
  - [Kolmogorov & Zabih 2002]

Depth-from-stereo has been investigated in computer vision for decades. Surprisingly, only relatively few algorithms have been proposed that can make efficient use of having more than 2 or 3 camera images of the scene available. Ideally, the depth estimation algorithm should meet a number of requirements. One approach that comes fairly close is based on combinatorial optimization.

Min-Cut/Max-Flow Optimization

SIGGRAPH2005

- Flow network
  - Directed graph
  - Weighted edges
    - Capacities x_i
  - Flow from source *s* to sinkl *t*
- Maximal Flow
  - Limited by “smallest edges"
  - [Ford&Fulkerson 1962], [Goldberg&Tarjan 1988]
    - $O(n^2m)$ (*n* nodes, *m* edges)
- s-t cut
  - Cut through limiting edges
  - Separation into two parts
    - Binary optimization
      - nodes v_i remain connect either to s or to t



The depth estimation algorithm we present is based on binary graph cuts. Consider a complex network of pipes of various different diameters and therefore different throughput, or capacity. What flows through this pipe network from one node, the source, to another node, the sink. How much water can maximally flow through the network ? Obviously, at the maximum flow some pipes are at the limit of their capacity while others could still transport more water. Given a network of different-capacity pipes, graph cut algorithms are able to determine which pipes are the limiting factor. Note that by cutting the network at these places, the graph is separated into to disjunctive parts, one still connected to the source, the other to the sink.

By finding a clever formulation of the depth-from-stereo problem in terms of a directed, weighted graph, depth can be reconstructed via binary graph-cut optimization.

Depth Reconstruction via Combinatorial Optimization

- For each pixels $p \in P_k$ in all cameras k
  - Assign label l
    - Finite number of different labels $l = (l_d, l_b)$
    - l_d depth level (multiple labels)
    - l_b foreground / background (binary label)
- Configuration λ
  - Label assignment of all pixels
  - Associated error

The underlying idea is to assign to each pixel one of finitely many different labels. Each label stands for a different depth layer, and in our case also for the pixel classification as image fore- or background. A configuration is a specific assignment of labels to all pixels. Each configuration corresponds either to a specific scene geometry, or to an impossible case (as we will see in a moment). By comparing the scene geometry as expressed by the label configuration to the all camera images, a quantitative measure can be established how likely this configuration may be. A likely configuration would have a small error value, while a very unlikely configuration would correspond to a high error value. Consequently, impossible configuration should have an infinitely high error value.

The task is now to find that label configuration that corresponds to the smallest possible error value.

α-Expansion

- Max-flow / Min-cut minimization
  - Binary optimization
  - Here: many different label
    - Multiway graph cut problem: NP-hard
- α -Expansion
  - [Boykov et al., PAMI 2001]
  - Iterative
    - Many binary optimizations in sequence
  - Initialization: allowed configuration
- Converges towards strong local minimum
    - Multiplicative factor to global minimum

If we had only two different labels to choose from, we could construct a graph whose source node represents one label and the sink node the other, while the other nodes would all correspond to all image pixels. All pixel nodes would be connected to the sink and source node, and more edges would link pixel nodes to represent potential correspondences. We would have to find a way to assign correct weights to the edges, but then the binary graph cut algorithm would separate the graph into two parts, with one part still attached to the source node label, and the other pixel nodes connected to the sink node label, and we would have our optimal configuration.

Unfortunately, we have more than two labels to choose from for each pixel. Because the multi-way graph cut problem is almost certainly NP-hard, we have to use an iterative strategy. Alpha expansion works by starting from a possible configuration and considering all labels subsequently. It iterates over all labels until no better configuration can be found anymore. While it cannot guarantee to converge towards the global minimum, it always finds a strong local minimum whose error is larger than the global minimum by at most a multiplicative factor that is determined by the error function.

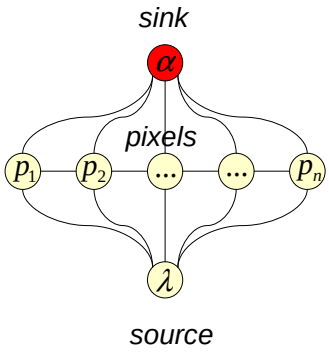
α- Expansion

- Iteration
  - Consider label α:
    - Find subset A of all pixels P such that:
      if α is assigned to A, total error becomes minimal
  - Repeatedly iterate over all possible labels
  - Terminate when total error decreases no further
- Determining A : binary problem !
  - Equivalent to minimal s-t cut

The alpha expansion algorithm considers one label after the other. For each currently considered label, a binary graph is constructed, and the graph cut is determined. Thus, the multi-way graph cut problem is attacked by performing many binary graph-cut optimizations.

Graph Construction

- Nodes correspond to pixels P
- Two special nodes
  - Source λ current label configuration
  - Sink α currently considered label
- Edge weights
  - Determined by energy functional
    - [Kolmogorov & Zabih, ECCV'02]
    - Additivity Theorem
      - Sum of two graph-representable functions is again graph-representable

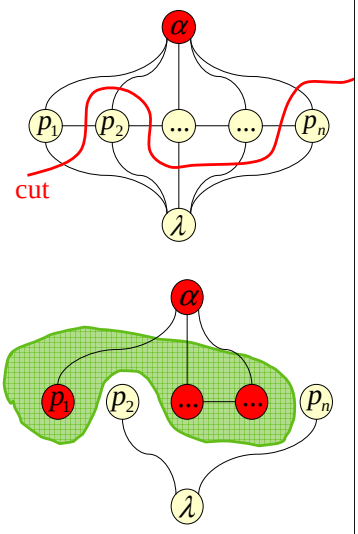


The graph is constructed as follows: The source node represents the current configuration, i.e., the individual label currently assigned to each pixel. The sink node is the label currently under scrutiny (e.g., the label for depth level 5, foreground). The edge weights are determined from an energy functional (that we will get to in a moment) in a clever way described by Kolmogorov and Zabih in their ECCV'02 paper.

Energy Minimization via Graph Cuts

SIGGRAPH2005

- Construct graph for λ, α
- Determine minimal s-t cut
  - Each pixel nodes remains connected either to the source or the sink
  - Sum of all cut edge weights is minimal
- Assign label α to all pixels that remain connect to the sink
- Repeatedly iterate α over all labels until no further energy minimization possible



The s-t cut of this graph is computed. All pixels whose nodes remain connected to the source node are not altered, while those pixels that remain attached to the sink node are assigned the current sink node label. Together, the labels of all pixels constitute the new configuration. For the next step, a new graph is constructed with the updated configuration as source node and a different label as sink node. The algorithm iterates until no more energy-minimizing graph cuts become possible.

Energy Functional

- Represents knowledge about image formation process
  - Concsistency, local similarity, Visiblity, Background
- Minimization: Find optimal configuration λ
  - Assign best-matching label to each pixel p,q∈ P

$$E(\lambda) = \sum_{p,q \in \mathrm{P}} \left(E^{p,q}_{photo}(\lambda) + bE^{p,q}_{smooth}(\lambda) + E^{p,q}_{vis}(\lambda) \right)$$

$$+ a \sum_{p \in \mathrm{P}} E^{p}_{background}(\lambda)$$

So far, we have not specified what makes a certain label configuration more (or less) probable than another given a set of multi-stereo images. This is the job of the energy functional. The energy functional represents our high-level knowledge about how the content of a set of images of a scene should behave. Our error functional consist of 4 terms that we are going to describe in more detail.

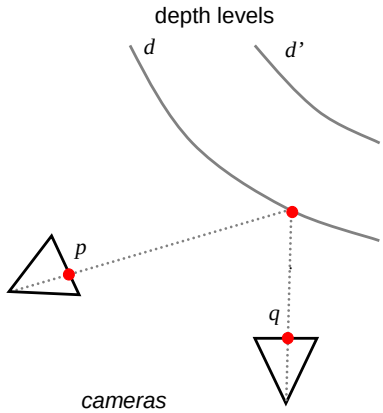
Photo-Consistency

$$\sum_{p,q\in P}\left(E^{p,q}_{photo}(\lambda)+E^{p,q}_{vis}(\lambda)+b\,E^{p,q}_{smooth}(\lambda)\right)+a\sum_{p\in P}E^{p}_{background}(\lambda)$$

SIGGRAPH2005

$$E^{p,q}_{photo}(\lambda):=\begin{cases}-C(p,q) & \text{if } \{\langle p,\lambda(p)\rangle,\langle q,\lambda(q)\rangle\}\in I\\ 0 & \text{else}\end{cases}$$

- Set I of interacting pixels $\{<p, l>,<q, l>\}$
  - $q\in P_k$, $p\notin P_k$ *from different cameras*
  - q *closest pixel in camera* k *to projection of* $<p, l>$
  - p,q *same label*
- *Projections of the same scene point*



The photo-consistency term represents our expectation that a scene point is similar in appearance in different images. Two pixels p and q from different cameras can only belong to the same scene point if they fulfill the epipolar constraint, i.e., p and q can potentially be the projections of the same 3D scene point, and if both pixels the same depth label l. All pixel pairs of a configuration that meet all three requirements form the set of interacting pixels. Reasonably, photo consistency can be measured for interacting pixels only, while for all other pixels the term is set to zero.

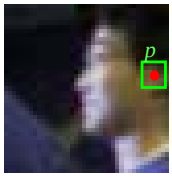
Photo-Consistency

$$\sum_{p,q\in P}\left(E^{p,q}_{photo}(\lambda)+E^{p,q}_{vis}(\lambda)+b\,E^{p,q}_{smooth}(\lambda)\right)+a\sum_{p\in P}E^{p}_{background}(\lambda)$$

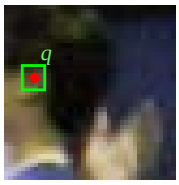
SIGGRAPH2005

$$E^{p,q}_{photo}(\lambda):=\begin{cases}-C(p,q) & \text{if } \{\langle p,\lambda(p)\rangle,\langle q,\lambda(q)\rangle\}\in I\\ 0 & \text{else}\end{cases}$$

- Similarity measure $C(p,q)$
  - Local neighborhood of p, q
  - Normalized cross correlation (NCC)
    - of color values
    - of 2. derivative
      - Laplacian filter
      - edges, corners



camera 1



camera 2

For the interacting pixel pairs, we have to quantify their similarity. One convenient way to do so is to compute the normalized cross correlation (NCC) between the local neighborhoods of pixels p and q. If the regions around p and q are similar, the NCC is close to 1, else it is smaller.

Besides matching relative pixel color values, it is a good idea to additionally emphasize correct matches of object edges and corners because our Human Visual System is very sensitive to image discontinuities. To do so, we compute the second derivative of our images by applying the Laplacian operator. In the Laplacian-filtered images, object edges and corners show up with high contrast. We calculate the NCC also between these high frequency-enhanced images and add it to the pixel-color NCC value.

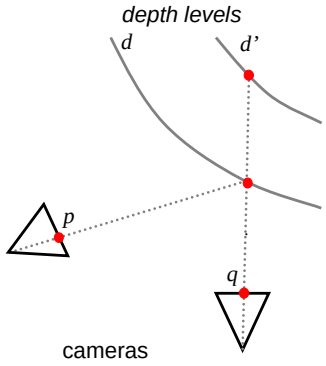
Occlusions

$$\sum_{p,q\in P}\left(E^{p,q}_{photo}(\lambda) + E^{p,q}_{vis}(\lambda) + b\, E^{p,q}_{smooth}(\lambda)\right) + a \sum_{p\in P} E^{p}_{background}(\lambda)$$

SIGGRAPH2005

$$E^{p,q}_{vis}(\lambda) := \begin{cases} \infty & \text{if } \{\langle p,\lambda(p)\rangle, \langle q,\lambda(q)\rangle\} \in O \\ 0 & \text{else} \end{cases}$$

- set O of mutually occluding pixels $\{<p, l>,<q, l'>\}$
  - $\{<p, l>,<q, l>\} \in I$ *corresponding pixels*
  - $l_d < l'_d$: $<p, l_d>$ *occludes* $<q, l'_d>$
    - *Impossible configuration*



When taking images of a three-dimensional scene from different viewpoints, parts of the scene are visible in some images but are occluded in other camera views. This causes some label assignments to be physically impossible

Our label-based discrete scene geometry representation allows us to elegantly determine which label assignments are not possible. In our example here, pixel p and q would be interacting pixels (i.e., correspond to the same 3D scene point at depth level d) if both were labeled l. However, pixel q is labeled l' which corresponds to a larger depth level d' (d'> d). But this cannot be because the scene point corresponding to pixel p occludes depth level d' from view for pixel q. Thus, <p,l> and <q, l'> is an impossible configuration which must be prevented by assigning an "infinitely" high energy to the visibility term. For possible configurations, the visibility term has no effect and is set to zero.

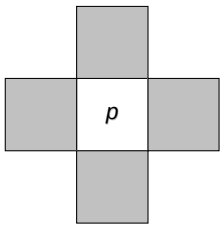
Smoothness

$$\sum_{p,q\in P}\left(E^{p,q}_{photo}(\lambda) + E^{p,q}_{vis}(\lambda) + b\, E^{p,q}_{smooth}(\lambda)\right) + a \sum_{p\in P} E^{p}_{background}(\lambda)$$

SIGGRAPH2005

$$E^{p,q}_{smooth}(\lambda) = V^{p,q}(l,l') := \begin{cases} 0 & \text{if } q \notin N_p \text{ or } l = l' \\ \text{else} & 2L_{\max} - \left\|\nabla^2 B(p)\right\| - \left\|\nabla^2 B(q)\right\| \end{cases}$$

- Depth discontinuity and segmentation along edges
  - p, q from same camera, neighbors
  - If not same depth, fore-/background:
    - Error measure dependent on second derivative



4-connectivity of pixel p: N_p

The photo-consistency term is able to determine pixel correspondences most robustly in image regions of high variations. In more or less uniform image areas, mismatches can occur. To minimize faulty correspondence assignments, the smoothness term punishes neighboring pixels p and q in smooth image regions if both pixels have different labels. Smoothness is quantified by the average of the second-derivative image over the patch around the pixels. If the average values is low, the image is locally smooth, and the smoothness term is assigned a high value (2 Lmax-…). The smoothness term is evaluated for each image pixel p with respect to his 4 neighboring pixels q.

Background Segmentation

$$\sum_{p,q\in P}\left(E^{p,q}_{photo}(\lambda) + E^{p,q}_{vis}(\lambda) + b\, E^{p,q}_{smooth}(\lambda)\right) + a \sum_{p\in P} E^{p}_{background}(\lambda)$$

SIGGRAPH2005

$$E^{p,q}_{background}(\lambda) := \begin{cases} C_b(p) & \text{if } \lambda(p)_b = \text{false} \\ \infty & \text{if } \lambda(p)_b = \text{true and } \lambda(p)_d \neq b_d(p) \\ 0 & \text{else} \end{cases}$$

- $C_b(p)$ *Normalized cross-correlation*
  - *local pixel color values*
    video image – background image
    - *Punish high similarity*
- *Impossible configuration*
  - *Pixel classified as background but different depth*
- *Uniform background*
  - *correlation possibly erroneous*

Finally, we want to make use of the static background images we took prior to recording the action in order to enhance depth estimation robustness. If a pixel p is assigned a label indicating that it belongs to the foreground, its local neighborhood is compared to the corresponding background image via the normalized cross correlation (NCC) C. By assigning the NCC value to the background term in this case, high similarity between the current image and the background image is punished, because it is likely that the pixel is actually depicting background. If the background region is uniform, the NCC value is not reliable, and we do not punish the pixel's foreground label. If the pixel is assigned a label indicating that it belongs to the background, but its depth level is not identical to the background depth, the configuration is not possible, resulting in an "infinitely" high error value. This implies, of course, that we have estimated background scene depth previously. In fact, we just apply the same estimation approach described here to the background images by leaving out this last background segmentation term.

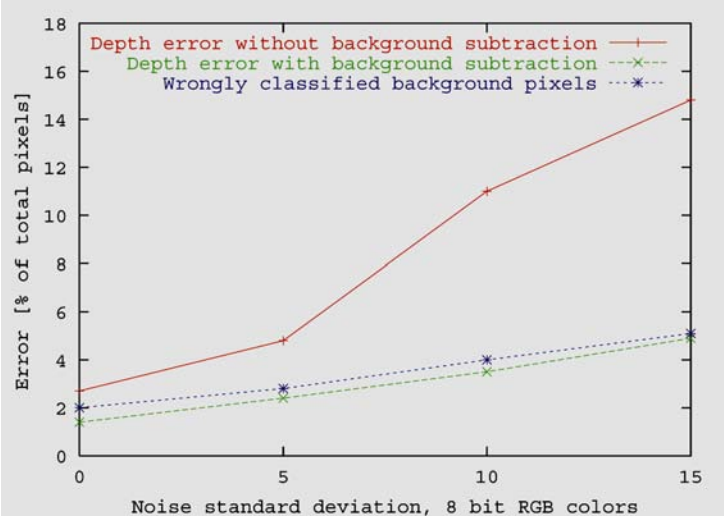
Foreground/Background Stereo Results

SIGGRAPH2005

- Static background images & depth
- Depth & classification for each pixel

The described algorithm returns dense depth maps and foreground segmentation masks for all recorded video images. The algorithm runs off-line. On a conventional 1.7GHz PC, it takes about 65 seconds to iterate once over 65 labels for four 320x240-pixel images.

Robustness



[Goldluecke & Magnor, CVPR'03]

The use of background images to simultaneously estimate depth and segment scene foreground increases reconstruction robustness considerably. To evaluate estimation robustness, we rendered a set of synthetic images, giving us ground-truth per-pixel depth and segmentation information. To the pixel values we added Gaussian noise of different sigma, stated in 8-bit pixel values along the x axis. The graph shows the percentage of pixels that are assigned erroneous labels. Without background images, error rate increases quickly with noise level. If background images are available, much more robust results are obtained. The amount of pixels assigned wrong depth or that are segmented wrongly is almost identical.

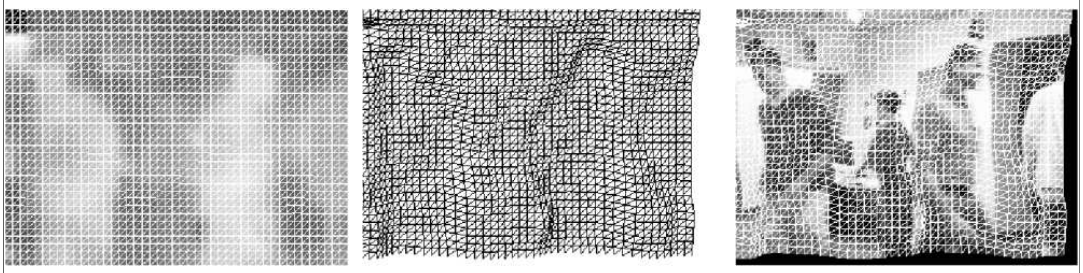
Dynamic Light Field Warping

- Light field / Lumigraph rendering
  - Arbitrary viewpoint outside convex hull
  - Random access to all images
    - Bandwidth problem
- 3D warping
  - Viewpoint on recording hull
    - In-between camera positions
  - 4 images per view
    - Disparity-compensate neighboring images
    - Blend warped images

Light field rendering and Lumigraph rendering re-sample the entire acquired image data to render any view from outside the scene's convex hull. However, the light field data must be randomly accessed. For very large data sets, such as dynamic light fields, local memory and/or bandwidth quickly limit the overall light field size.

To achieve interactive rendering frame rates while streaming the dynamic light field from hard drive, we restrict the rendering viewpoint to lie on the recording hull spanned by all camera positions. In practice, this is not a major limitation as the scene is typically best resolved from viewpoints at distance similar to that of the recording cameras (see also [Zitnick et al., Siggraph'04]). For each rendered view, four images from camera positions closest to the desired viewpoint are taken into account.

Warping & Blending on Graphics Hardware

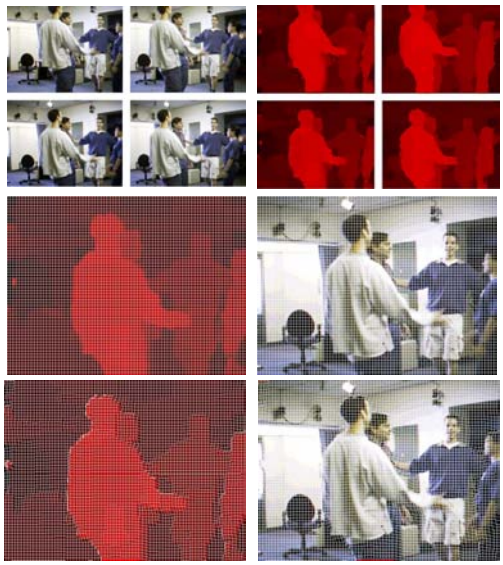


- Project regular triangle mesh onto depth map
- Warp mesh to viewpoint
- Projective texture-map triangle mesh
- Weight images according to distance from viewpoint

Dynamic light field warping can be implemented entirely on graphics hardware. The four dynamic light field images are uploaded as reference images to the graphics board as textures. For each image, a triangle or quad mesh is rendered whose vertices are assigned the depth values of the pixel labels they correspond to. The (x,y) vertex coordinates represent the mesh's texture coordinates. The mesh is rendered from the desired viewpoint which automatically warps the mesh to correctly display the varying depths. The mesh is then projectively textured. The renderings from all four reference images are weightedly blended together according to spatial distance of the viewpoint from the respective camera positions.

Interactive Rendering

- Per time frame
  - 4 images + depth maps
- Warping
- Blending
- Visibility
  - Depth buffer
- Sharp contours
  - Fore- / Background

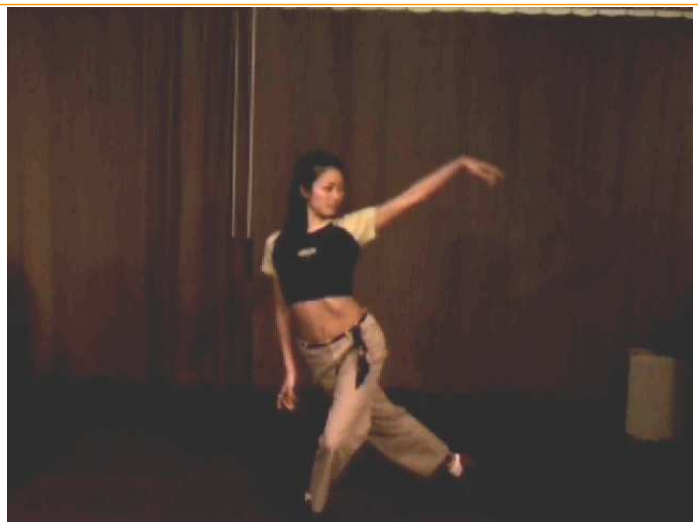


Per time frame, four images and depth-label maps are retrieved from hard drive. The images are uploaded to texture memory, while the depth maps are used to generate the warping meshes. Mesh resolution can be varied to trade off rendering speed vs. disparity compensation accuracy. Occlusions are accounted for by using the depth test.

To increase rendering quality, the output image can be rendered in two passes. First, only the static background is rendered, before in a second pass the foreground is rendered. For enhanced rendering results, the edges along the foreground can be alpha-blended using an additional opacity mask [Zitnick et al, Siggraph'04].

“Camera Hopping”

SIGGRAPH2005



Frozen frame (24 cameras)

Stanford Multi-Camera Array

If simply hopping from camera to camera, the difference in image parallax is annoyingly visible.

Disparity-compensated Rendering

640x480, 30 fps (PC + GeForce3)

[Goldluecke & Magnor, VMV'02]

In contrast, smooth transitions between camera views are obtained if dynamic light field warping is applied.

Wide-baseline VBR
9:45 – 10:15

Wojciech Matusik

Marc Pollefeys

Overview

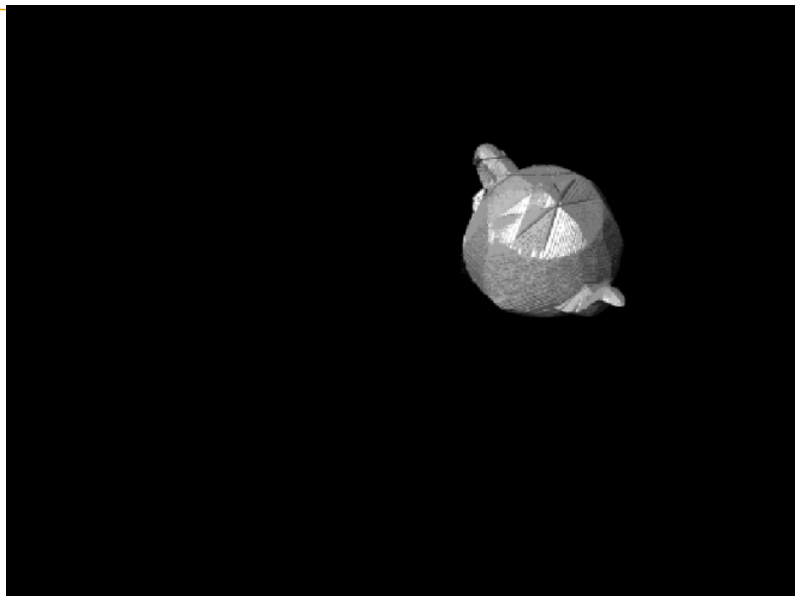
Visualizing and navigating within virtual environments composed of both real and synthetic objects has been a long-standing goal of computer graphics. The term “Virtualized Reality™”, as popularized by Kanade, describes a setting where a real-world scene is “captured” by a collection of cameras and then viewed through a virtual camera, as if the scene was a synthetic computer graphics environment. In practice, this goal has been difficult to achieve. Previous attempts have employed a wide range of computer vision algorithms to extract an explicit geometric model of the desired scene. We present algorithms for synthesizing virtual renderings of real-world scenes in real time. Not only is our technique fast, it also makes few simplifying assumptions and has few restrictions.

Previous Work

- Visual Hull
  (Laurentini'94)
- Volume Carving
  (Potmesil'87, Szeliski'93, Seitz'97)
- View-Dependent Texture Mapping
  (Debevec'96, Debevec'98, Buehler'01)
- Virtualized Reality
  (Rander'97, Kanade'97, Narayanan'98)

Many researchers have used silhouette information to distinguish regions of 3D space where an object is and is not present. The ultimate result of this carving is a shape called the object's *visual hull*. A visual hull always contains the object. Moreover, it is an equal or tighter fit than the object's convex hull. A common method used to convert silhouette contours into visual hulls is volume carving. This method removes unoccupied regions from an explicit volumetric representation. All voxels falling outside of the projected silhouette cone of a given view are eliminated from the volume. This process is repeated for each reference image. The resulting volume is a quantized representation of the visual hull according to the given volumetric grid. Kanade's virtualized reality system is perhaps closest in spirit to the rendering system that we envision. Their initial implementations have used a collection of cameras in conjunction with multi-baseline stereo techniques to extract models of dynamic scenes.

What is a Visual Hull?

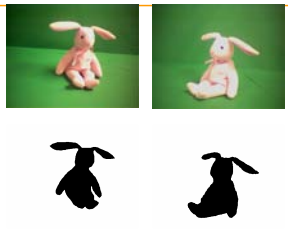


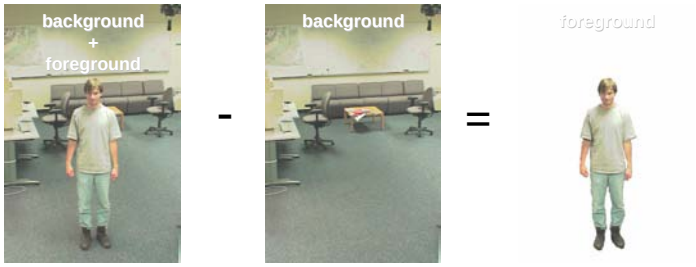
Laurentini introduced the *visual hull* concept to describe the maximal volume that reproduces the silhouettes of an object. Strictly, the visual hull is the maximal volume constructed from all possible silhouettes. We compute the visual hull of an object with respect to a finite number of silhouettes. The silhouette seen by a pinhole camera determines a three-dimensional volume that originates from the camera's center of projection and extends infinitely while passing through the silhouette's contour on the image plane. We call this volume a silhouette cone. All silhouette cones exhibit the hull property in that they contain the actual geometry that produced the silhouette. For our purposes, a visual hull is defined as the three-dimensional intersection of silhouette cones from a set of pinhole silhouette images.

Why use a Visual Hull?

SIGGRAPH2005

- Can be computed robustly
- Can be computed efficiently





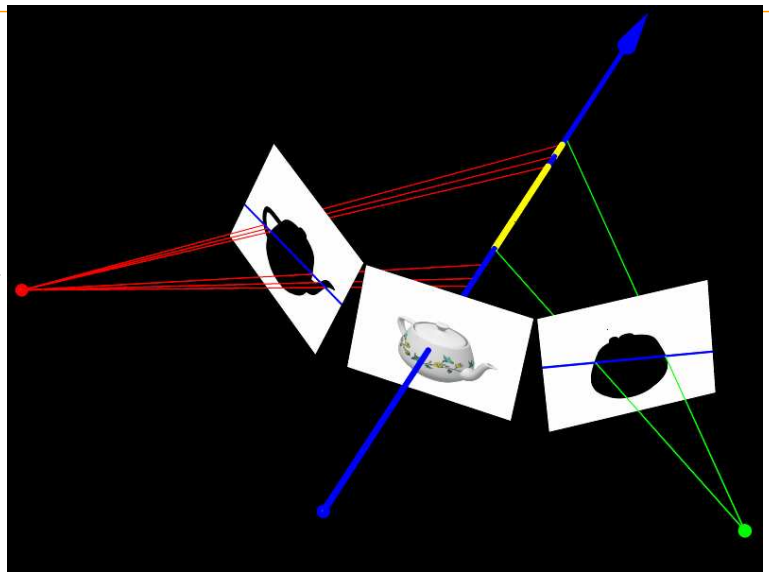
We use the visual hull because silhouettes can be obtained robustly using well known methods such as background subtraction or blue-screen matting. The second reason is that visual hulls can be rendered efficiently.

Image-Based vs. Polyhedral VH

	IBVH	PVH
Output	**point samples**	**triangles**
Complexity	**linear in # cameras**	**quadratic in # cameras**
Rendering	**software**	**hardware**
Visibility	**per-pixel**	**per-triangle**
Performance	**8 frames/sec**	**15 frames/sec**

There are a few major differences between IBVHs and PVHs. The major difference is shape representation. IBVH outputs surfels. PVH outputs triangles. Complexity is linear with the number of cameras for IBVH and quadratic for PVH Rendering is done in software for IBVH and in hardware for PVH. Visibility is computed per surfel for IBVH and per triangle for PVH. In early 2001, we could get 8 fps for IBVH and 15 fps for PVH. Today, these performances should be a few times faster.

Image-Based Visual Hull

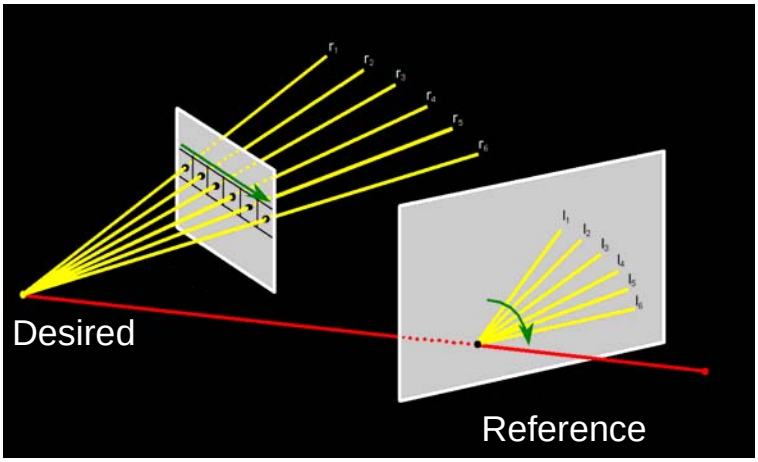


IBVH algorithm: As depicted here, much of the computation can be done in the 2d image domain. For a particular desired viewing ray (shown in blue), we compute its projection onto the reference views as line segments shown here in red. In each of these reference views, we compute the intersection of the projected segment with the 2d silhouette. The resulting intervals can be lifted back into 3D to the viewing ray. The intersection of all computed intervals gives the visual hull sample. Efficient computation requires us to be able to quickly intersect the projected rays with the 2d silhouette representation.

Observation

SIGGRAPH2005

- Incremental computation along scanlines

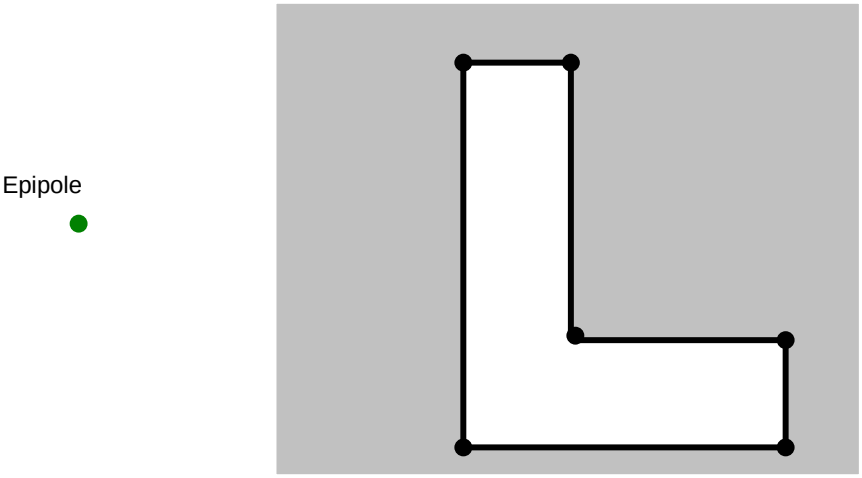


The main observation needed to make this step efficient is that as one traverses the desired pixels along a scanline, the projected viewing rays in the reference views trace out a pencil of rays with monotonically increasing angles around the epipole. This will allow us to efficiently compute line/silhouette intersection efficiently.

Binning

SIGGRAPH2005

- Sort silhouette edges into bins

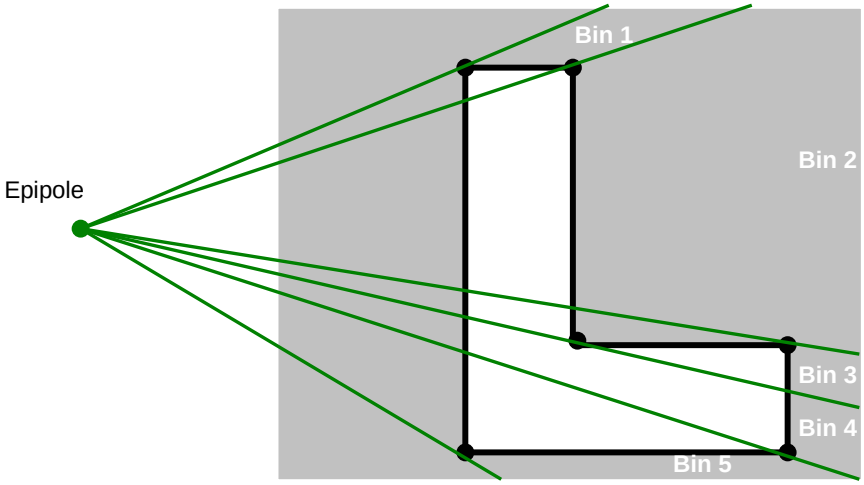


In each reference view, we first break the image domain up into a set of bins. Each silhouette vertex defines a bin boundary. Each silhouette edge is placed in all bins it falls in.

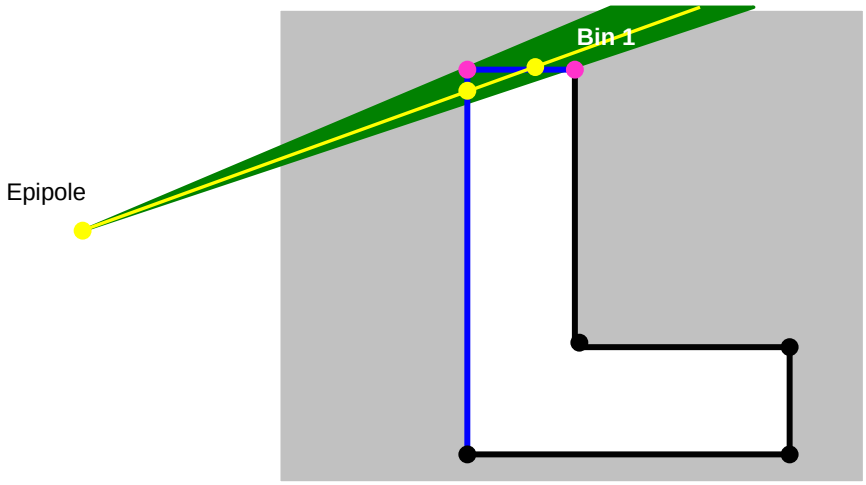
Binning

SIGGRAPH2005

- Sort silhouette edges into bins

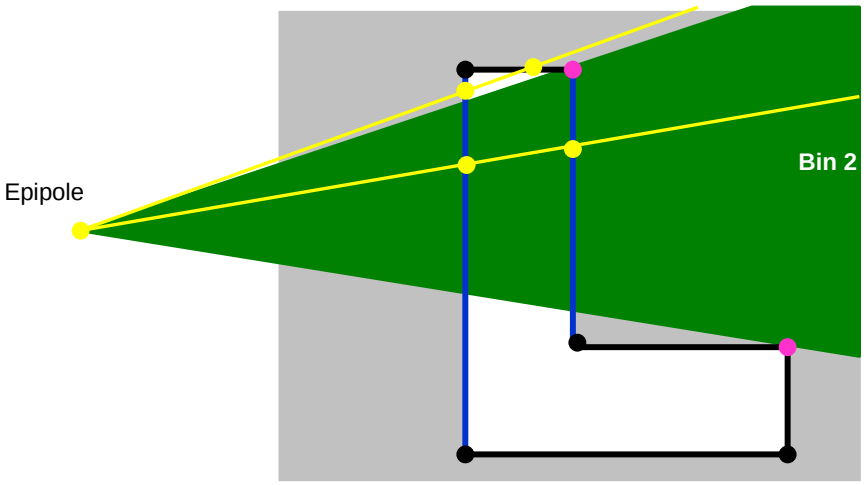


Scanning

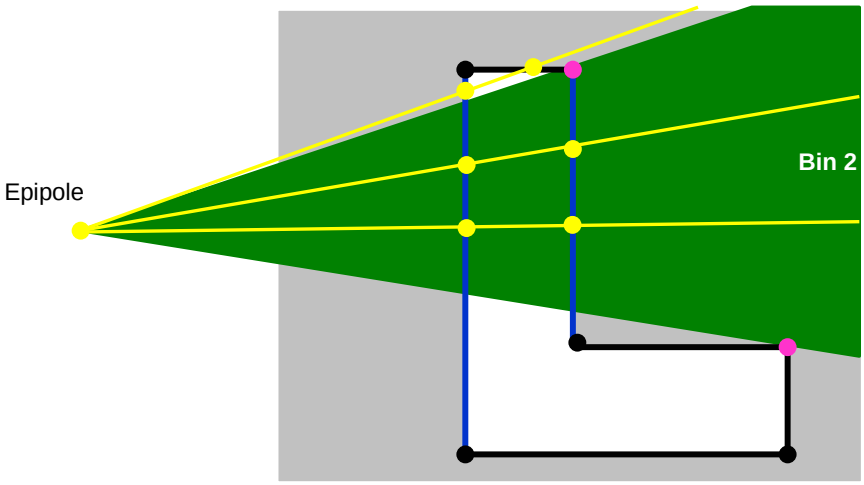


We then traverse the pixels in each scanline of the desired view. As we trace across the scanline, we simultaneously track the projected view ray as it moves across the bins. Thus, for each projected view ray, we only need to test a small number of silhouette edges for intersection.

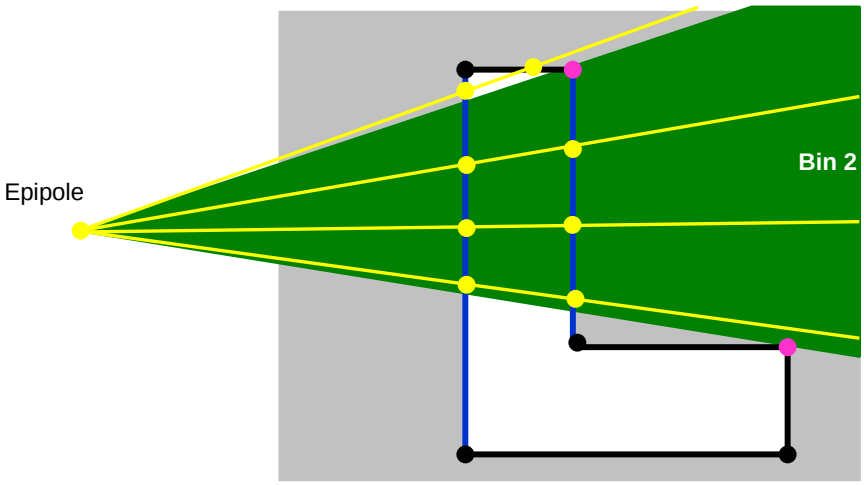
Scanning



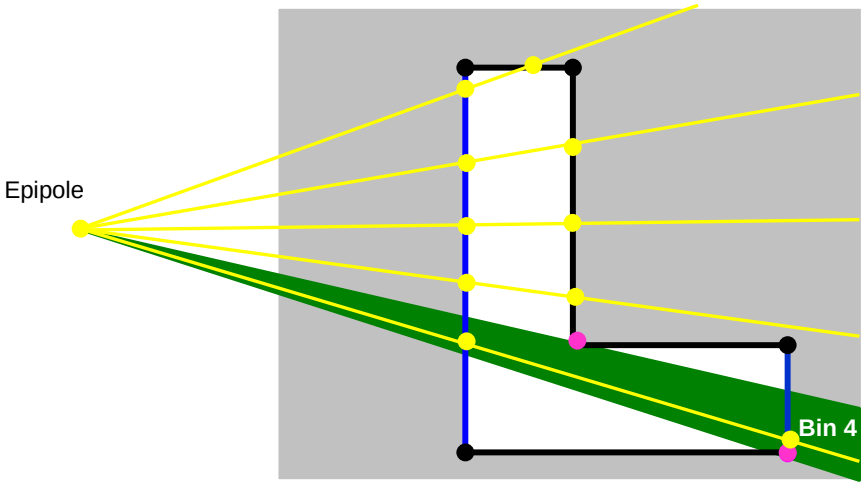
Scanning



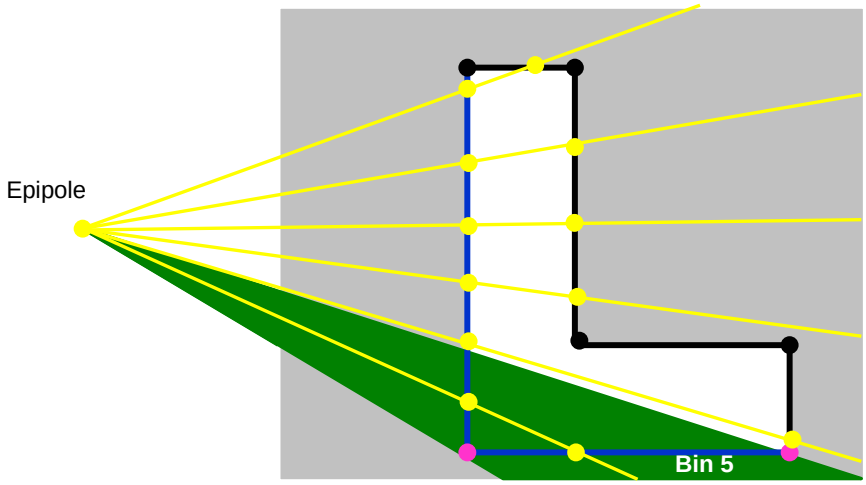
Scanning



Scanning



Scanning

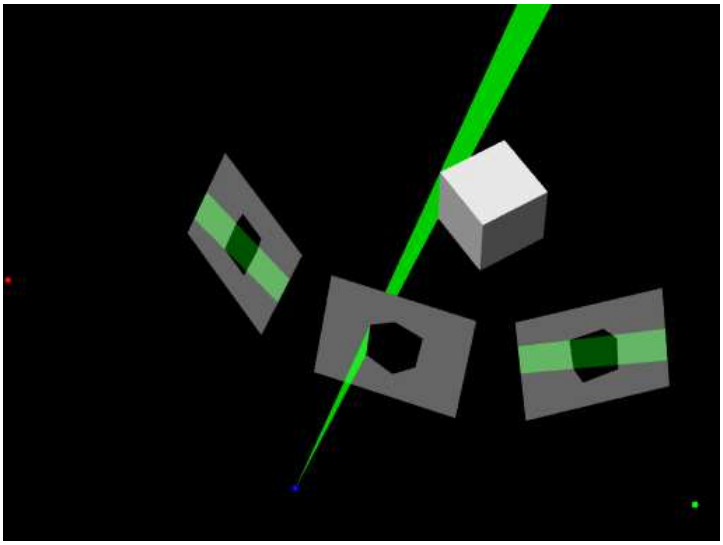


IBVH Shape

SIGGRAPH2005

The images are computed at 400x400 resolution. The colors represent distance to the camera (white – close, black – far). The external contours of the computed model look smooth and do not exhibit quantization artifacts characteristic to the volumetric approaches.

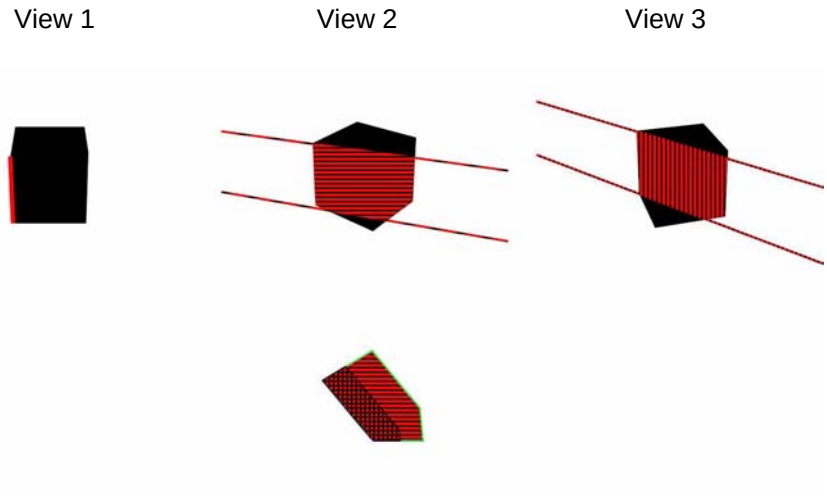
Polyhedral Visual Hulls



An algorithm to compute polyhedral visual hulls: In order to compute the visual hull with respect to the input silhouettes, we need to compute the intersection of the cones defined by the input silhouettes. The resulting polyhedron is described by all of its faces. Note that the faces of this polyhedron can only lie on the faces of the original cones, and the faces of the original cones are defined by the projection matrices and the edges in the input silhouettes.Thus, a simple algorithm for computing the visual hull might do the following: For each input silhouette s_i and for each edge e in the input silhouette s_i we compute the face of the cone. Then we intersect this face with the cones of all other input silhouettes. The result of these intersections is a set of polygons that define the surface of the visual hull.

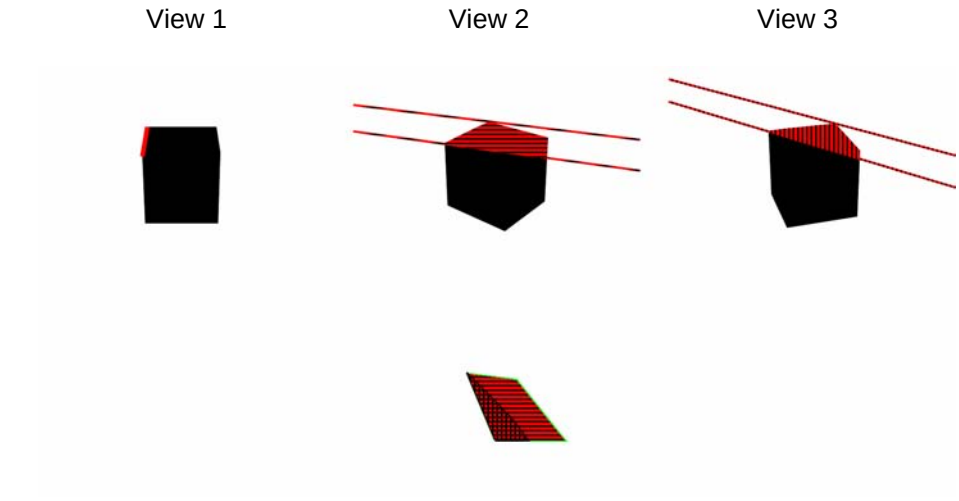
Reduction to 2D: The intersection of a face of a cone with other cones is a 3D operation (these are polygon-polyhedron intersections). We observe that these intersections can be reduced to simpler intersections in 2D. This is because each of the silhouette cones has a fixed scaled cross-section; that is, it is defined by a 2D silhouette. Reduction to 2D also allows for less complex 2D data structures to accelerate the intersections. To compute the intersection of a face f of a cone $cone(s_i)$ with a cone $cone(s_j)$, we project f onto the image plane of silhouette s_j. Then we compute the intersection of projected face f with silhouette s_j. Finally, we project back the resulting polygons onto the plane of face f.

Computing the PVH



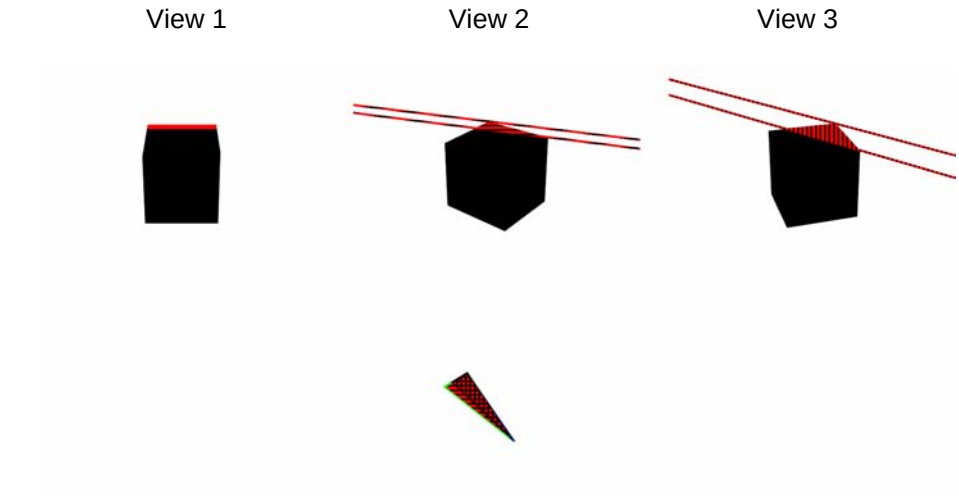
We demonstrate the procedure with a simple example. Here we have three silhouettes of a cube. First, we are going to compute surface due to view 1. We process each silhouette edge for view 1. We project each extruded edge onto view 2 & 3. We compute the intersections of the projected extruded edge with silhouette 2 & 3. We lift the intersections back to 3D and compute their intersection.

Computing the PVH



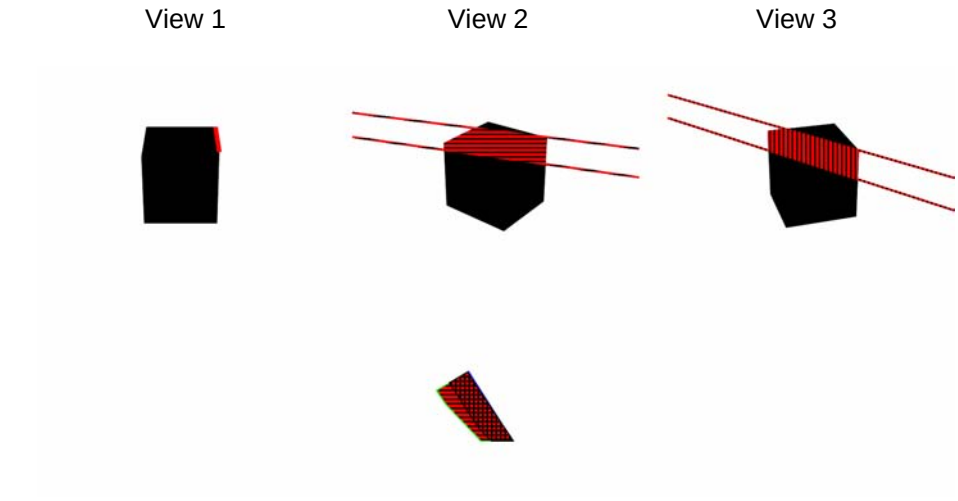
We process 2nd silhouette edge of view 1.

Computing the PVH



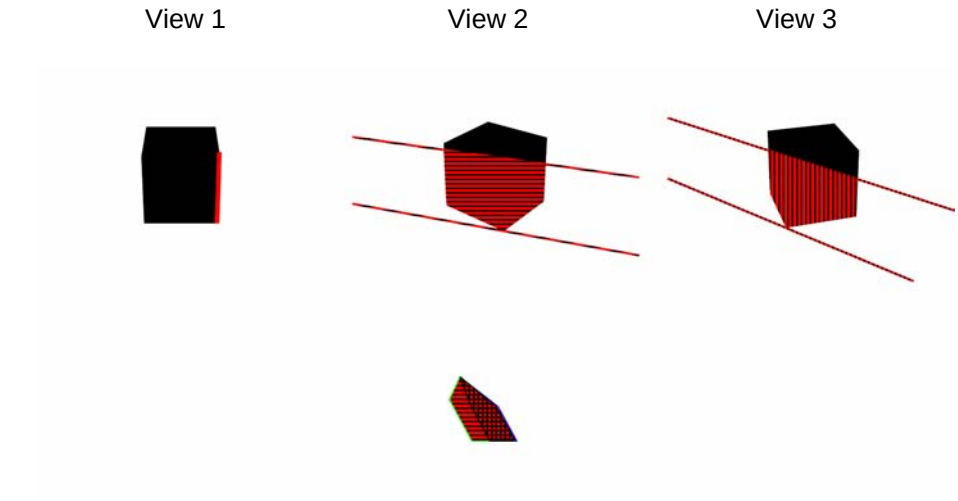
We process 3rd silhouette edge of view 1.

Computing the PVH



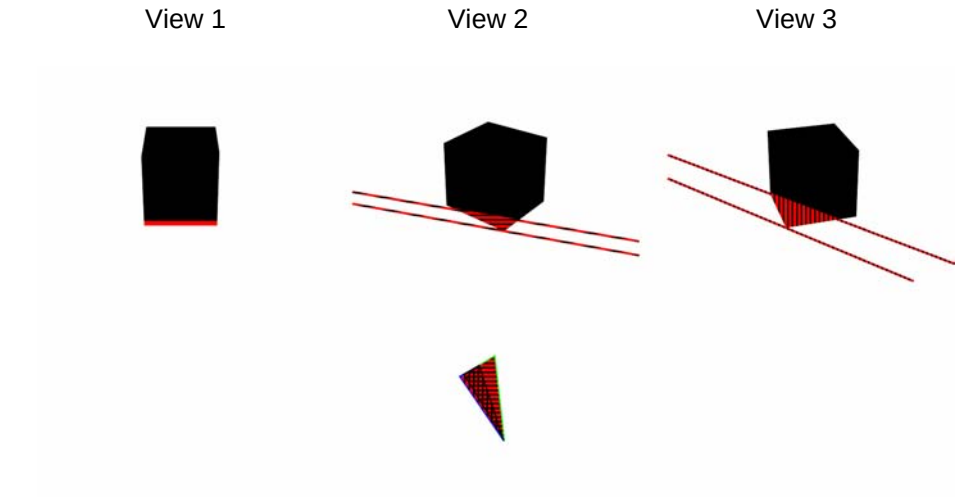
We process 4th silhouette edge of view 1.

Computing the PVH



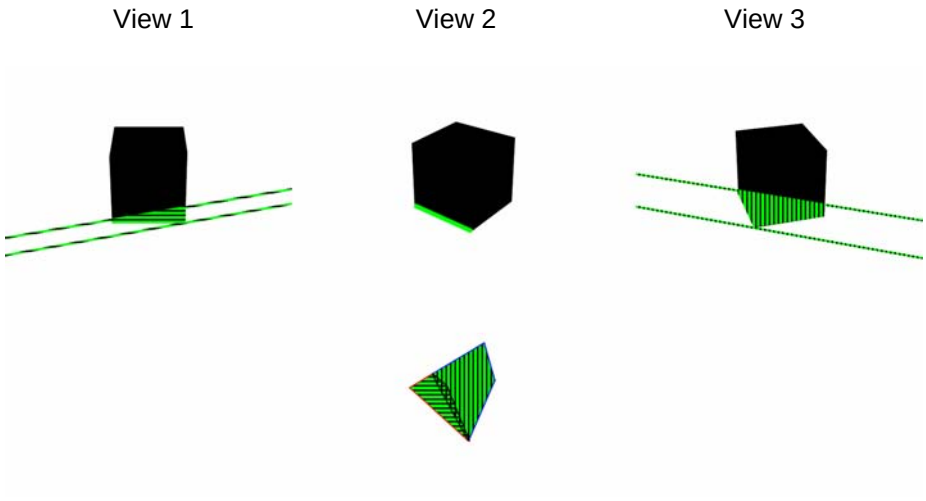
We process 5th silhouette edge of view 1.

Computing the PVH



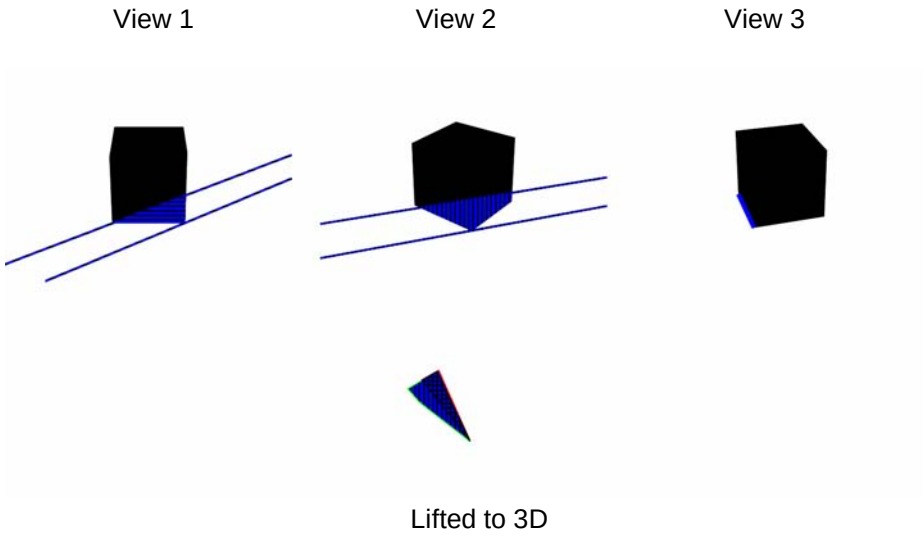
We process 6th silhouette edge of view 1.

Computing the PVH



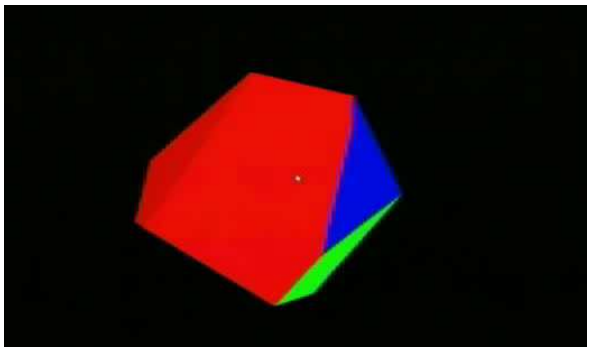
Similarly we process all silhouette edges of view 2.

Computing the PVH



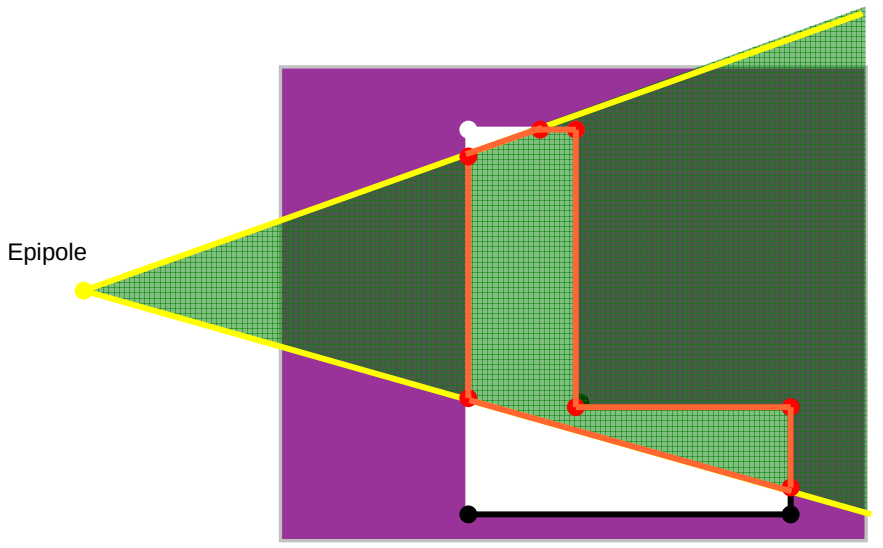
Similarly we process all silhouette edges of view 3.

Final Result



And this is the final result of the intersection in 3D. The colors red, green, and blue correspond to each reference view.

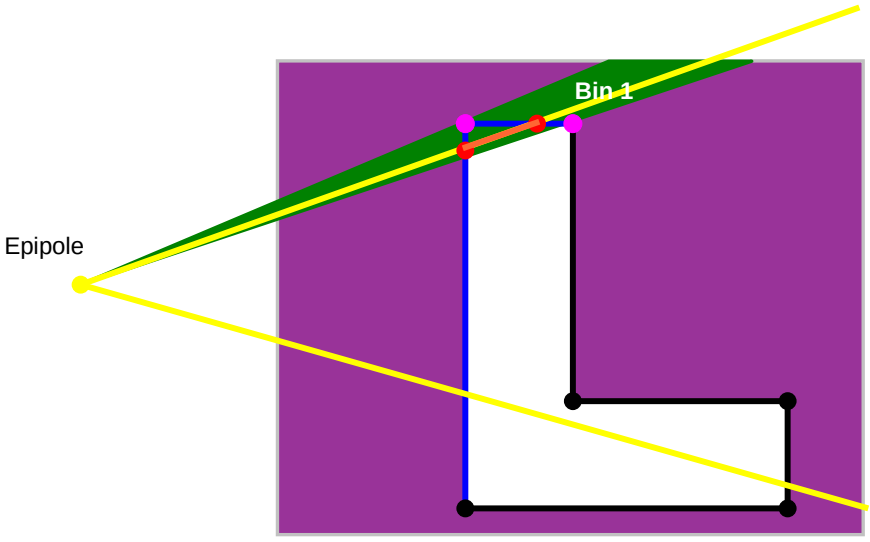
Acceleration Using Edge-Bins



Efficient Intersection of the Projected Cone Faces with a Silhouette. Using the edge bin data structure, we can compute efficiently the intersection of the projected cone c_i with the silhouette s_j of some other cone c_j . In order to compute the intersection we process the faces of cone c_i in consecutive order. We start by projecting the first face f_1 onto the plane of silhouette s_j . The projected face f_1 is defined by its boundary lines with the values α_{p1}, α_{p2}. First, we need to find a bin $b = \{\alpha_{start}, \alpha_{end}, S\}$ (S is a set of edges in bin b) such that $\alpha_{p1} \in (\alpha_{start}, \alpha_{end})$. Then, we intersect the line α_{p1} with all the edges in S. Since the edges in S are sorted based on the increasing distance from the projected vertex of cone c_i we can immediately compute the edges of the resulting intersection that lie on line α_{p1}. Next, we traverse the bins in the direction of the value α_{p2}. As we move across the bins we build the intersection polygons by adding the vertices that define the bins. When we get to the bin $b = \{\alpha_{start}, \alpha_{end}, S\}$ such that $\alpha_{p2} \in (\alpha_{start}, \alpha_{end})$ we intersect the line α_{p2} with all edges in S and compute the remaining edges of the resulting polygons. It is important to note that the next projected face f_2 is defined by the boundary lines α_{p2}, α_{p3}. Therefore, we do not have to search for the bin α_{p2} falls into. In this manner we compute the intersection of all projected faces of cone c_i with the silhouette s_j.

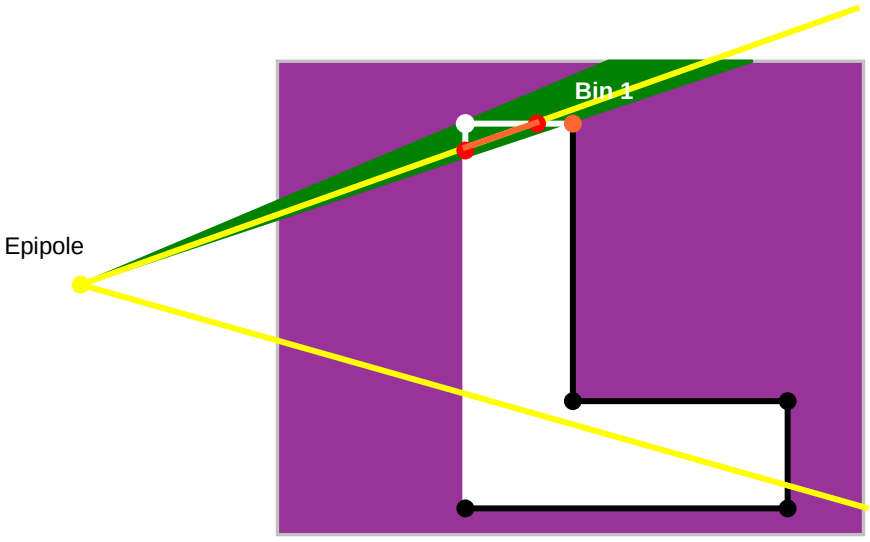
Example: Given the projected edge (yellow) and a sample silhouette, we would like to compute the intersection(orange) efficiently.

Acceleration Using Edge-Bins

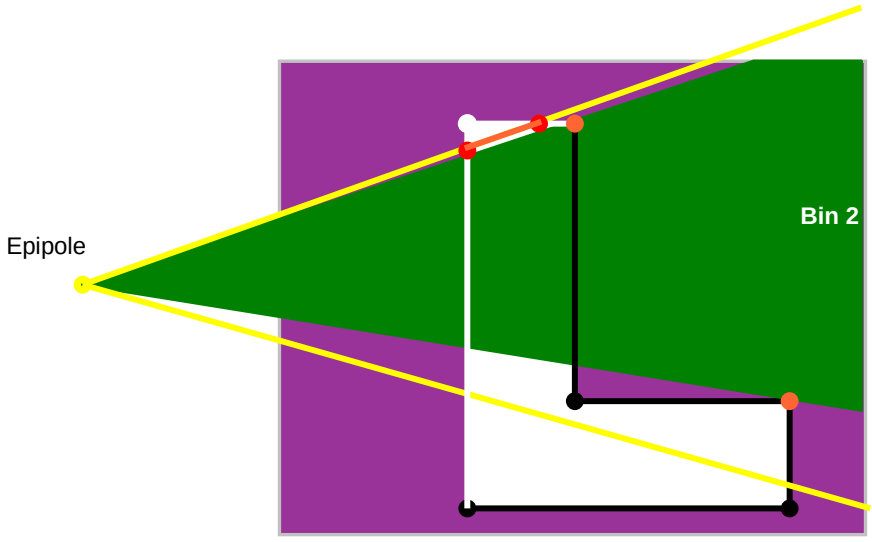


First, we are going to find the bin of the start of the projected extruded edge and intersect it with all edges in the bin.

Acceleration Using Edge-Bins

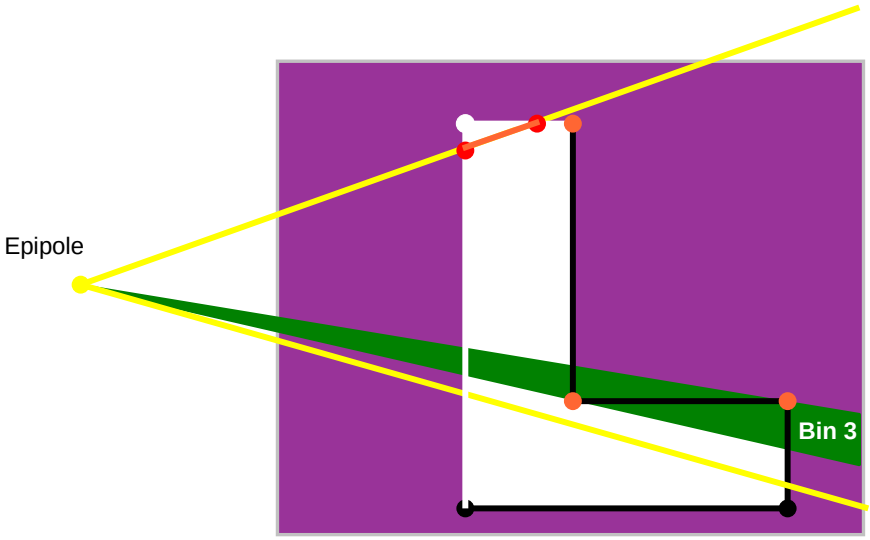


Acceleration Using Edge-Bins

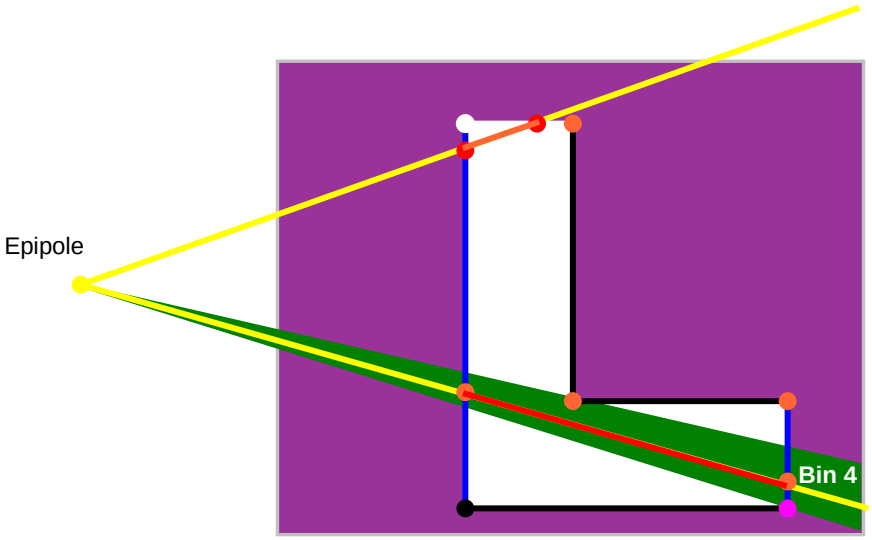


Next, we are going to traverse all bins towards the other extent of the projected extruded edge and add all vertices/edges in the bins.

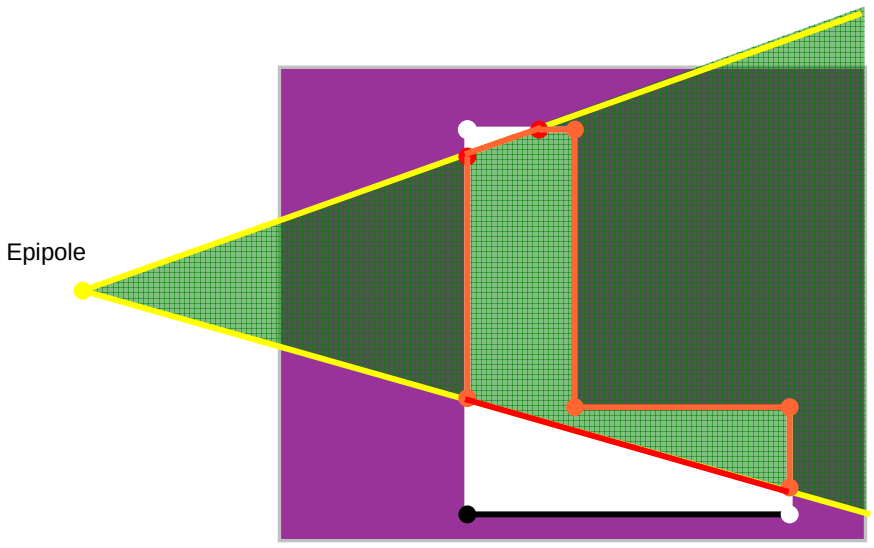
Acceleration Using Edge-Bins



Acceleration Using Edge-Bins



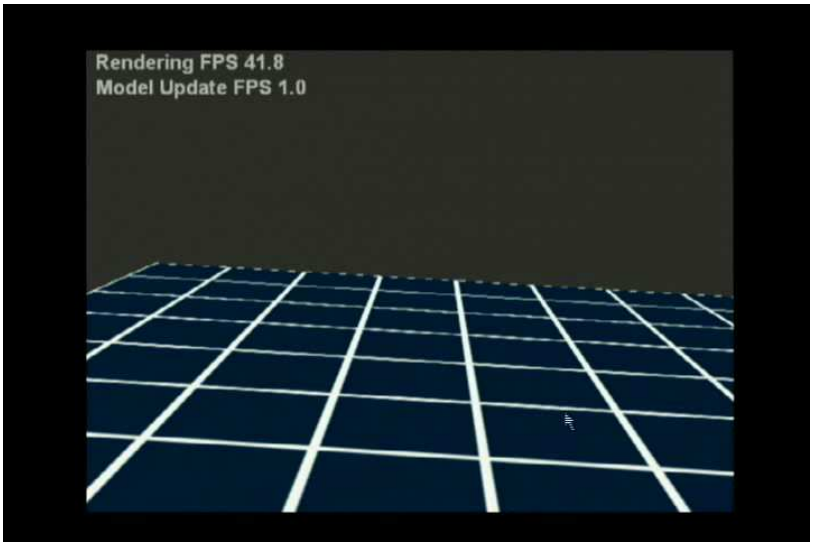
Acceleration Using Edge-Bins



In this way, we can compute this intersection efficiently. Note that, now we do not need to search for the first bin of the next projected extruded edge.

This bin is exactly the same as where we ended this intersection.

PVH Shape



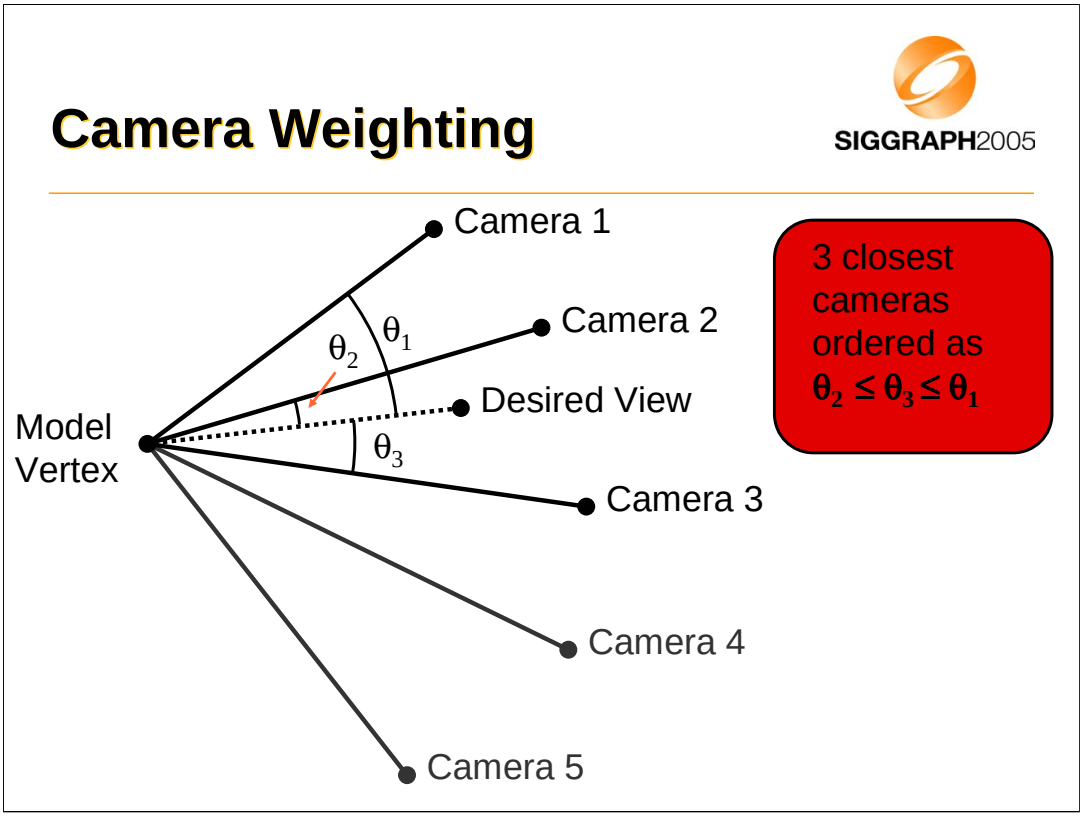
Our system computes polyhedral visual hull models at a peak 15 frames per second, which is the frame rate at which our cameras run. The rendering algorithm is decoupled from the model construction, and it can run up to 30 frames per second depending on the model complexity. The actual frame rates of both components, especially rendering, are dependent on the model complexity, which in turn depends on the complexity of the input silhouette contours. In order to maintain a relatively constant frame rate, we simplify the input silhouettes with a coarser polygonal approximation. The amount of simplification is controlled by the current performance of the system. We show flat-shaded renderings of a polyhedral visual hull that was captured in real-time from our system. These images demonstrate the typical models that our system produces. The main sources of error in creating these models is poor image segmentation and a small number of input images.

View-Dependent Rendering

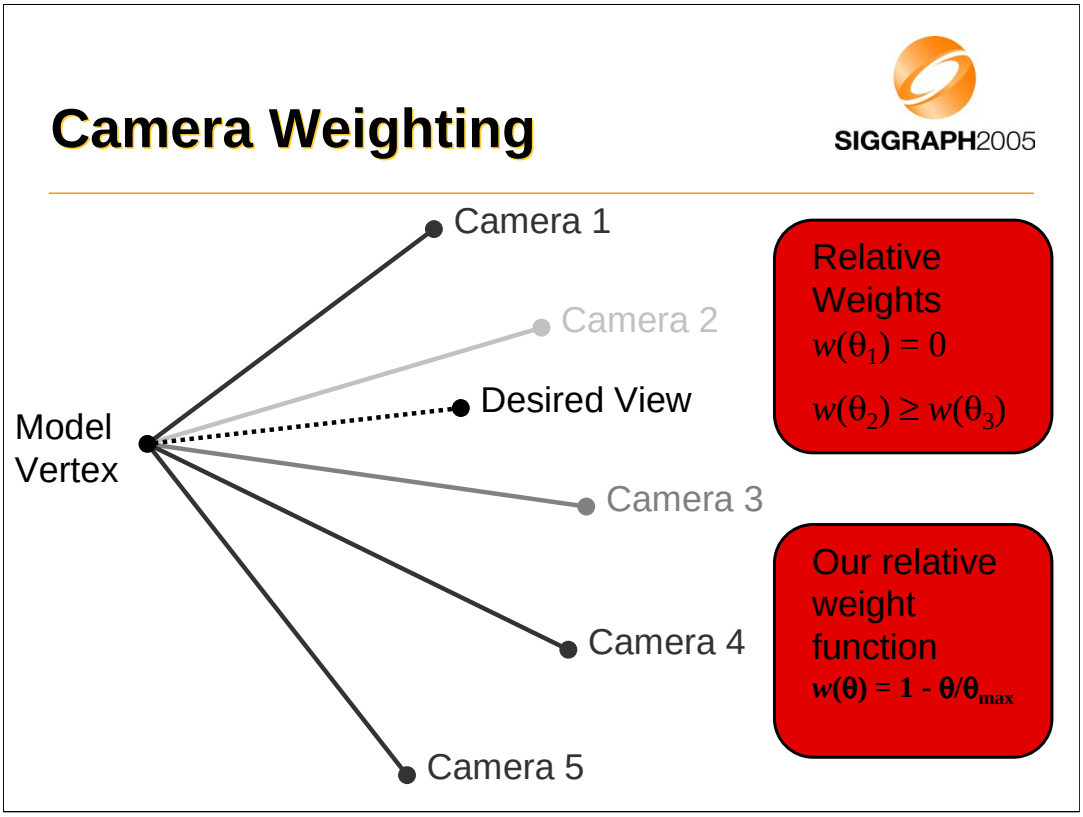
SIGGRAPH2005

- Camera weighting factors are computed at each
  - pixel (IBVH)
  - vertex (PVH)
- Visibility information is incorporated

We can greatly improve the appearance of the visual hull model by texturing it with the video images. We use a type of view-dependent texture mapping for this purpose. In our approach, each surface element is textured with images from multiple cameras. The images are blended together according to camera weighting factors.The weighting factors are computed at surface element for IBVH and at each vertex of the model for PVH. With a small number of cameras, visibility is an important consideration. That is, we don't want to texture a polygon with a camera that did not see that surface point.



This is how we compute the camera weighting factors at each vertex. I'll explain the process for a single vertex of the model. Let's assume five cameras view the vertex. We wish to determine view-dependent camera weighting factors based on a desired view. The approach that we use is based on the approach in the Unstructured Lumigraph Rendering. This approach is designed for continuous blending of images from cameras that are arranged in any position. First, we find a fixed number of cameras closest to the desired one. Closeness is measured by the angular difference between the desired viewing ray and the camera viewing rays. In this case, we've chosen the 3 closest cameras, although you can choose 2 or more. The cameras that are not chosen (cameras 4 and 5) are assigned weights of zero.

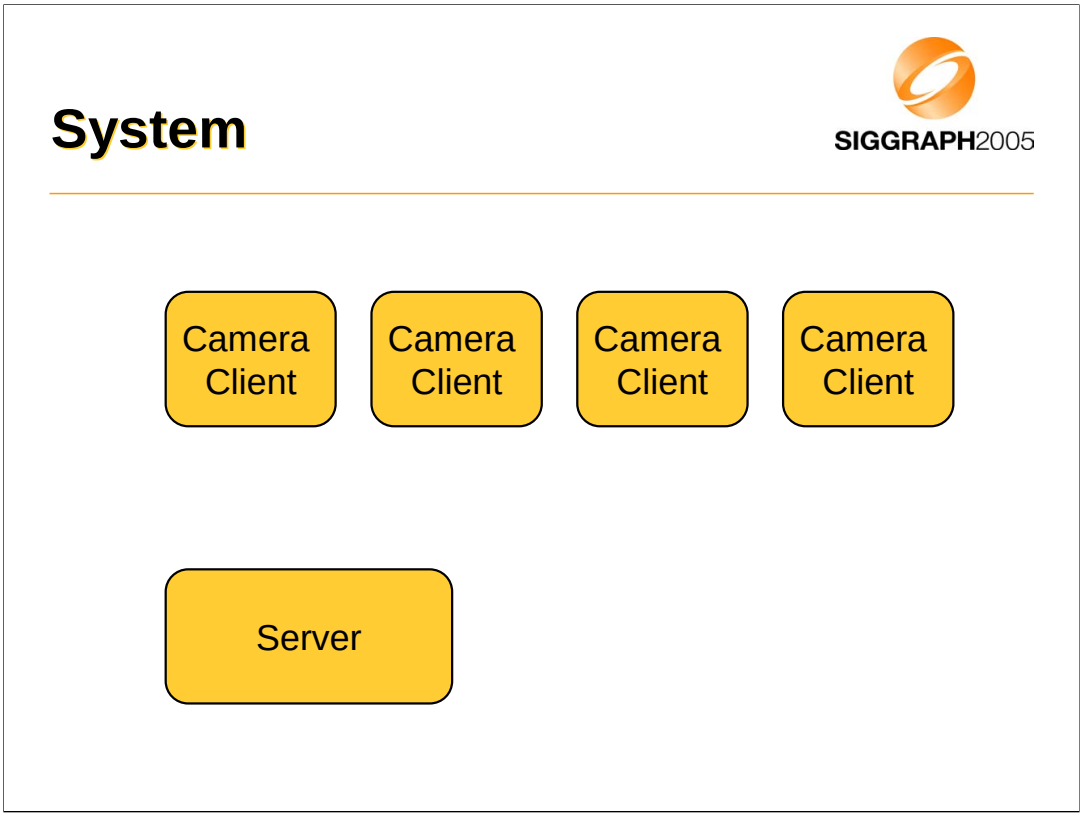


Next we compute relative weights for the three chosen cameras. To maintain continuity, weights are chosen such that the weight of the farthest camera (camera 1) is zero. On the other hand, a maximum weight is assigned to cameras whose viewing rays coincide with the desired viewing ray (this case does not occur in this example). Weights monotonically decrease from the maximum to zero. In this case, camera 2 has the largest weight and camera 3 the next largest. For use in rendering, the relative weights are renormalized so that they add to one. In our system, we use the particularly simple relative weight function that is shown here. It has a maximum value of 1 when the desired viewing ray coincides with a camera viewing ray, and it falls off linearly to zero at the farthest camera angle.

Results

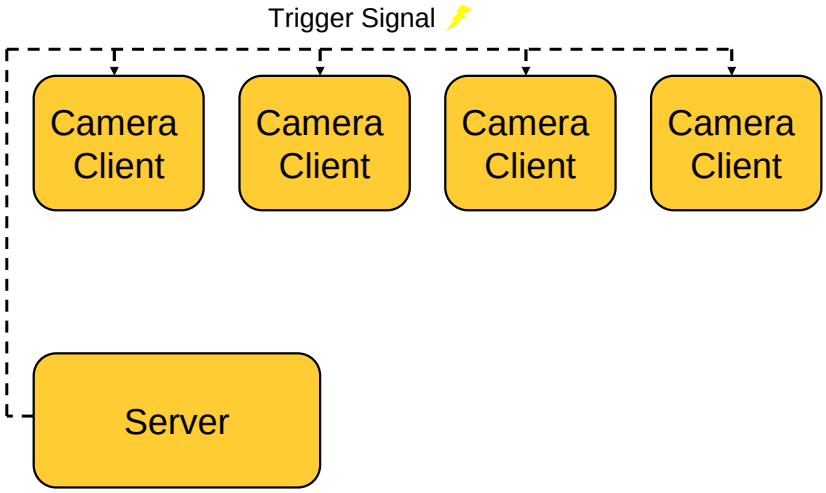
Interpolating the camera weights over the entire model results in a blending field.We can visualize the blending field by assigning each camera a single color, and then blending these colors according to the blending field. Here is a blending field in which the 4 source cameras have been assigned the colors red, green, blue, and yellow.

System



The current system uses four calibrated Sony DFW-V500 IEEE-1394 video cameras. Each camera is attached to a separate client (600 MHz Athlon desktop PC). The cameras are synchronized to each other using an external trigger signal. Each client captures the video stream at 15 fps and performs the following processing steps: First, it segments out the foreground object using background subtraction. Then, the silhouette and texture information are compressed and sent over a 100Mb/s network to a central server. The central server (2x933MHz Pentium III PC) performs the majority of the computations. The server application has the following three threads: • *Network Thread* - receives and decompresses the textures and silhouettes from the clients. • *Construction Thread* - computes the silhouette simplification, volume intersection, and visibility. • *Rendering Thread* - performs the view-dependent texturing and display. Each thread runs independently of the others. This allows us to efficiently utilize the multiple processors of the server. It also enables us to render the visual hull at a faster rate than we compute it. As a result, end users perceive a higher frame rate than that at which the model is actually updated.

System


Trigger Signal
Camera Client
Camera Client
Camera Client
Camera Client
Server

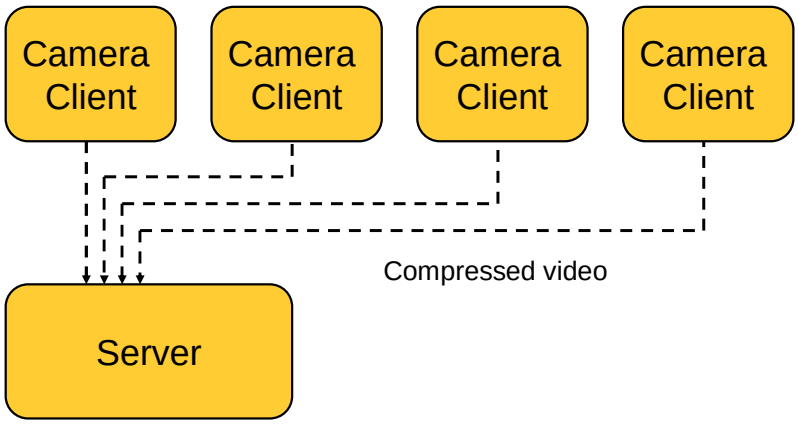
System



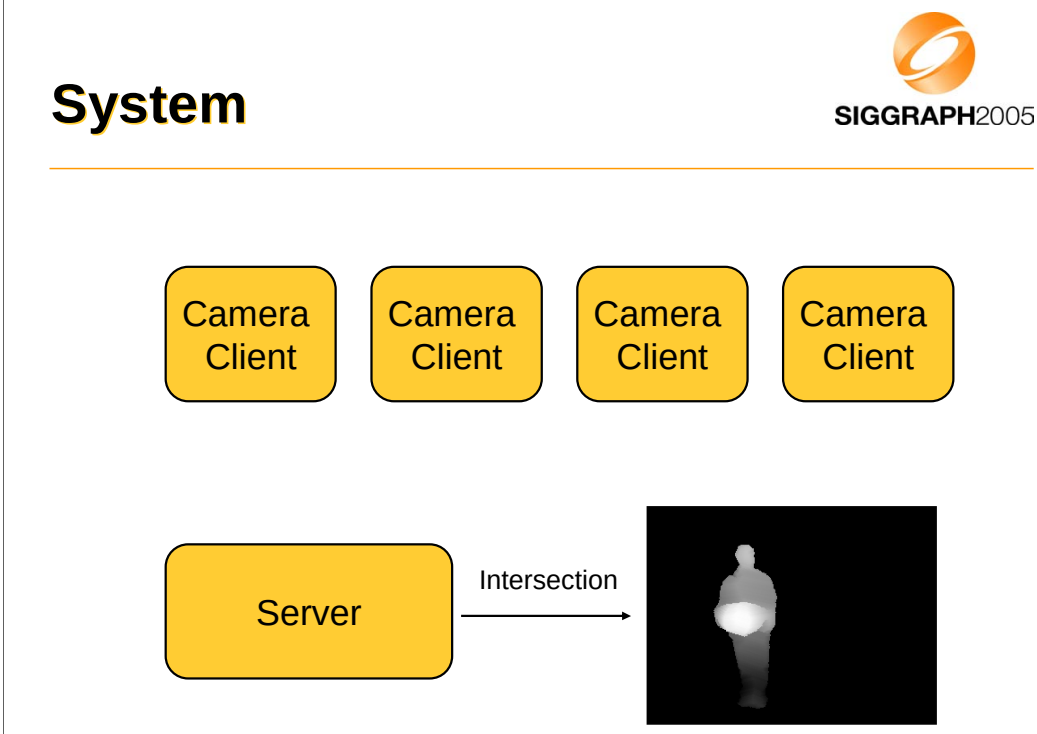

Camera Client | Camera Client | Camera Client | Camera Client

Server

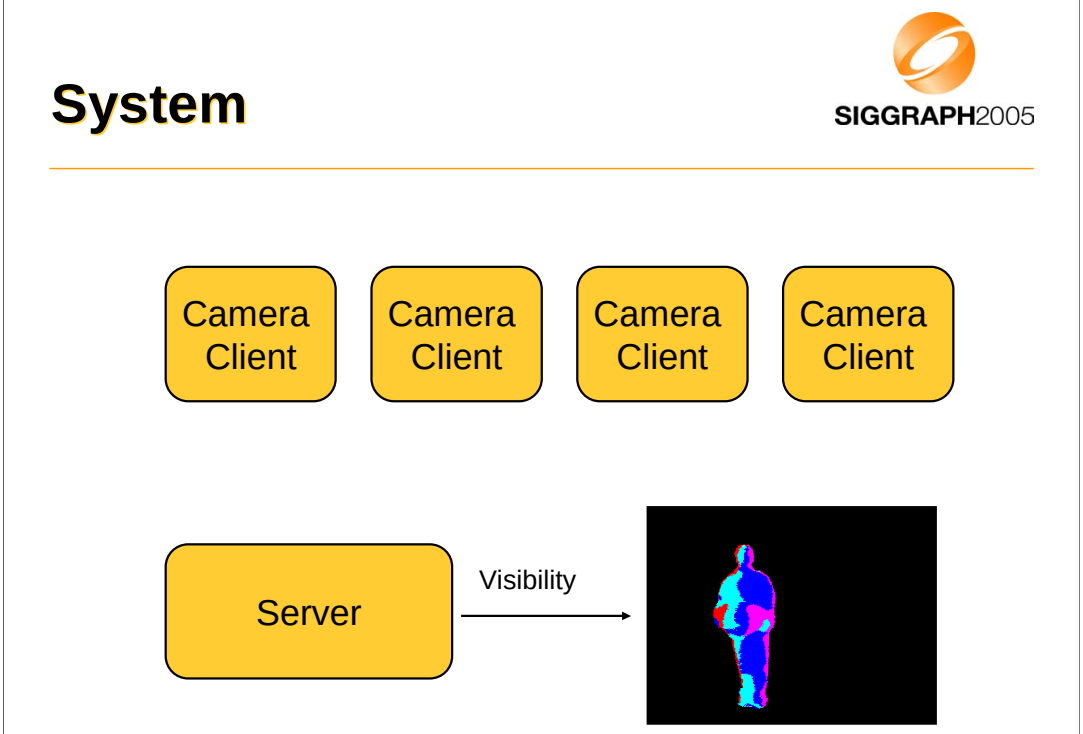
System


Camera Client
Camera Client
Camera Client
Camera Client
Compressed video
Server

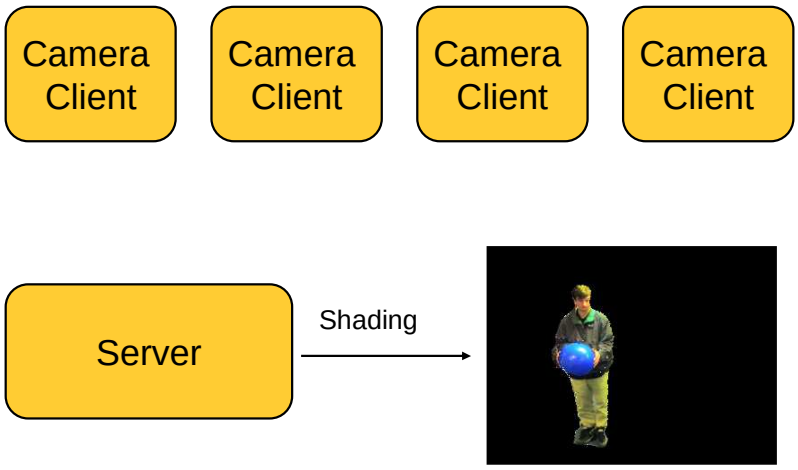
System


Camera Client
Camera Client
Camera Client
Camera Client
Server
Intersection

System


Camera Client
Camera Client
Camera Client
Camera Client
Server
Visibility

System


Camera Client
Camera Client
Camera Client
Camera Client
Server
Shading

Conclusions

- Visual hulls can be computed robustly and efficiently
- Visual hulls with view-dependent textures are convincing representations

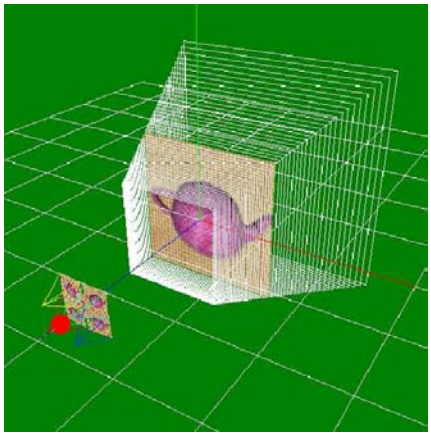
Visual hulls can be computed robustly and efficiently. I have demonstrated this showing two different algorithms. The visual hull provides a useful model whose combination of accurate silhouettes and textures provides surprisingly effective renderings that are difficult to distinguish from a more exact model.

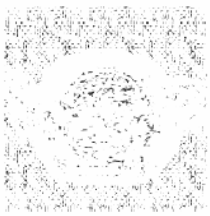
Photo-consistency

SIGGRAPH2005

- Plane-sweep multi-view depth estimation

[Collins, CVPR'96]





One of the constraints to estimate depth from multiple images is photo-consistency. This constraints expresses the fact that pixels that correspond to a point on the true object surface should have a consistent appearance, i.e. in general have the same color. A simple and efficient approach to compute depth based on this principle is the plane-sweep approach. The approach consist of sweeping a plane through the volume under consideration. For each location of the plane, all the images are projected on the plane and the algorithm verifies how similar pixels projected from different images are. Along rays corresponding to pixel of a reference view, the best match is chosen as the depth estimate for that pixel.

In the slide an example is shown that corresponds to a virtual thea pot in front of a canvas. On the top-right the images corresponding to the four cameras are blended together, on the bottom-right the corresponding sum-of-absolute-differences between RGB values is shown. Black corresponds to a perfect match.

Stereo on GPU

SIGGRAPH2005

[Yang and Pollefeys, CVPR'03]
[Yang and Pollefeys, JRI'05]

(Demo ATI 9800)

580M disparity hypothesis/sec (ATI X800)
e.g. 512x512x70disparities at 30Hz

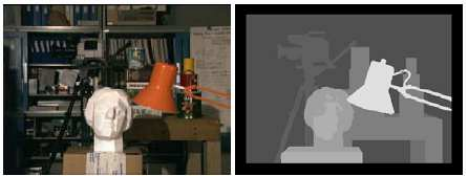
When only two views are being considered, photo-consistency corresponds to stereo computations. There has been a lot of emphasis on this special case in the area of computer vision and many algorithms have been proposed. Those vary from fast real-time algorithms that perform an independent per-pixel computation of the depth/disparity to algorithms that try to estimate a global optimal solution over the whole image.

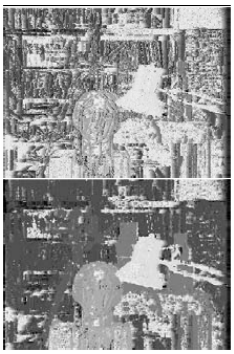
Here we show results from a fast real-time algorithm that runs on the GPU (Graphics Processing Unit).

Matching cost aggregation

SIGGRAPH2005

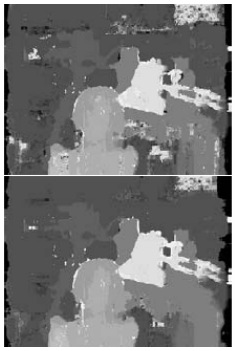
- Fast multi-resolution mipmap aggregation





(1x1)

(1x1+2x2)



(1x1+2x2 +4x4+8x8)

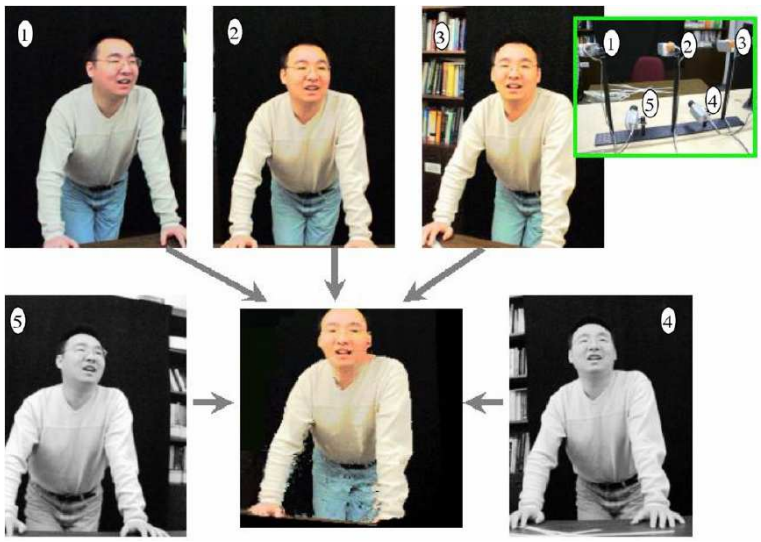
(1x1+2x2 +4x4+8x8 +16x16)

Because by only considering a single pixel at a time results are often ambiguous, especially when only two images are considered, typically the absolute difference is summed over a small neighborhood of pixels, assuming that neighboring pixels will mostly be at the same depth. For our real-time algorithm implemented on the GPU we use the build-in MipMap box filter to approximate the sum over a number of different neighborhood sizes. One can see how the results improve as more MipMap levels are included in the computation.

Novel view synthesis

SIGGRAPH2005

[Yang et al., PG'02]



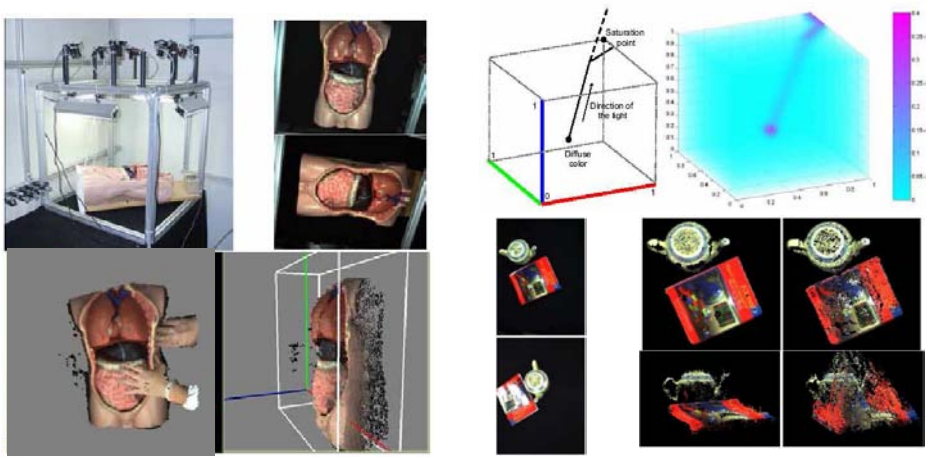
In fact, in stead of computing depth, this approach can also immediately be used to for novel view synthesis. In this case the reference view corresponds to the desired novel view and the color for a pixel is chosen as the consensus color from the multiple input cameras (i.e. the one yielding the smallest sum-of-absolute-differences).

Dealing with specular highlights

SIGGRAPH2005

[Yang, Pollefeys and Welch, ICCV'03]

- Extend photo-consistency to include highlights



The traditional approach to photo-consistency assumes that to be photo-consistent pixels must have the same RGB values. While this is true for Lambertian/diffuse surfaces (assuming the cameras have been photometrically calibrated), this is not true in general. In particular, for specular surfaces one can expect to have a mix between the diffuse RGB component of the surface reflection and the specular reflection of the light color. Assuming the color of the light is white, this means that and RGB value that corresponds to (Rd,Gd,Bd)+S.(1,1,1) would also be photo-consistent (with (Rd,Gd,Bd) being the diffuse color) and S the amount of specular reflection. In practice one also has to take saturation into account as shown on the upper right of the slide.

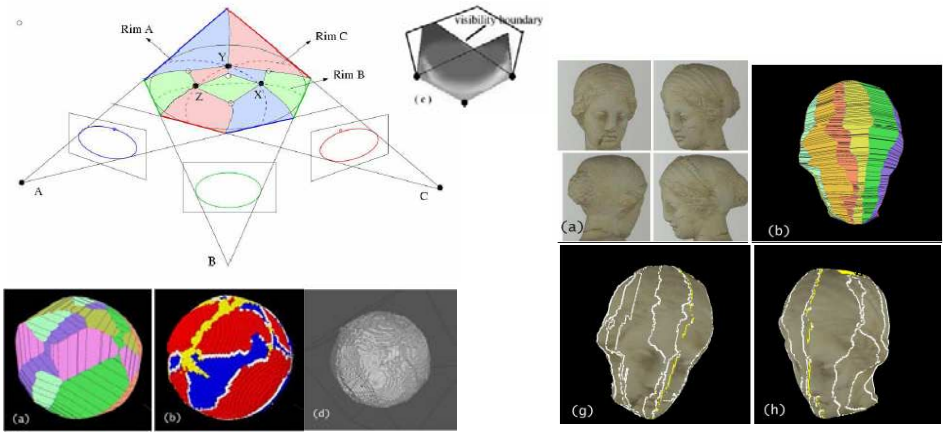
When using this new photo-consistency measure, it becomes possible to perform space-carving on specular surfaces, even in the presence of large area light-sources. A few examples are shown on the slide. The reconstruction example of the theapot + openGl book is shown with and without the use of our extended photo-consistency measure.

Silhouettes + photo-consistency

SIGGRAPH2005

[Sinha and Pollefeys, submitted ICCV'05]

- Global graph-cut for exact silhouettes and trade-off between photo-consistency and smoothness



Silhouette and photo-consistency constraints are complementary. Ideally, both constraints would be used together in a single optimization for the object shape. Several approaches have been proposed that attempt to integrate both approaches. In this slide we illustrate a global graph-cut based approach. The graph is constructed so that each valid cut represents a surface that exactly satisfies the silhouette constraints. The minimal cut then yields the best possible trade-off between photo-consistency and smoothness. A few examples are given of the initial visual hull that is used for the graph construction and then the resulting shape after graph-cut optimization.

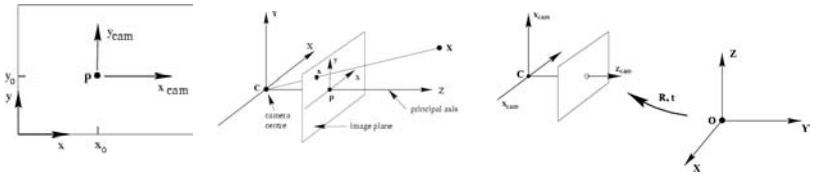
Break
10:15 – 10:30

Camera Calibration
10:30 – 11:00

Marc Pollefeys

Pinhole camera model

SIGGRAPH2005



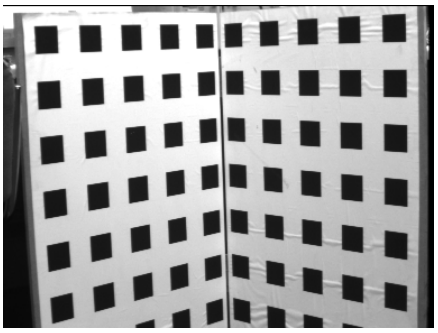
$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix}}_{\mathbf{K}} \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\lambda \mathbf{x} = \mathbf{P}\mathbf{X}$$

linear projection in homogeneous coordinates!

On this slide we introduce the pinhole camera model. This model is appropriate for perspective cameras and is a good approximation for most real camera. 3D points and 2D points are represented by homogeneous 4-vectors respectively 3-vectors, lambda is a non-zero scale-factor. A 3D point X is first transformed to camera coordinates through a Euclidean transformation. Then the point is projected to a normalized 2D point. This point which is equivalent to the incoming ray is then transformed by the calibration matrix K to obtain pixel coordinates. The calibration matrix K contains the intrinsic camera parameters. f_x and f_y represent the focal length measured in the width resp. height of pixels and (p_x,p_y) correspond to the location of the principal point. For photocameras $f_x=f_y$, but for video cameras (which are typically not rotated on their side during acquisition), this is not necessarily the case. All those effects can be combined in a single 3x4 projection matrix.

Simple camera calibration



need 6 or more known points
(not all on a single plane!)

$$\lambda \mathbf{x} = \mathbf{P}\mathbf{X}$$

$$\begin{cases} \lambda x &= \mathrm{P}_1\mathrm{X} \\ \lambda y &= \mathrm{P}_2\mathrm{X} \\ \lambda &= \mathrm{P}_3\mathrm{X} \end{cases}$$

$$\begin{cases} \mathrm{P}_1\mathrm{X} - x\mathrm{P}_3\mathrm{X} &= 0 \\ \mathrm{P}_2\mathrm{X} - y\mathrm{P}_3\mathrm{X} &= 0 \end{cases}$$

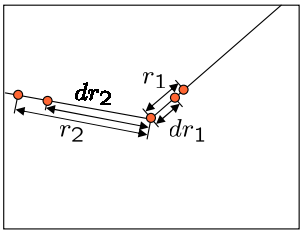
$$\begin{bmatrix} \mathrm{X}^\top & 0 & -x\mathrm{X}^\top \\ 0 & \mathrm{X}^\top & -y\mathrm{X} \end{bmatrix} \mathrm{p} = 0$$

$$\mathrm{p} = [\ \mathrm{P}_1 \quad \mathrm{P}_2 \quad \mathrm{P}_3\ ]^\top$$

(fig. from Hartley & Zisserman)

The traditional camera calibration technique consists of recording a picture of a know object. Each known correspondence between 2D image coordinates and 3D world coordinates provide 2 linear equations on the projection matrix. In general 6 points are sufficient to yield a unique solution (as long as at least 2 of them are not coplanar with the others).

Radial distortion



$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix} \mathcal{R} \left(\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right] \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \right)$$

$$\mathcal{R}\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right) = \begin{bmatrix} x \\ y \\ d^{-1} \end{bmatrix} \qquad d = 1 + \kappa_1(x^2 + y^2) + \kappa_2(x^2 + y^2)^2 + \ldots$$

(fig. from Hartley & Zisserman)

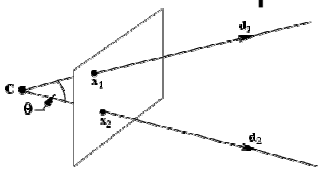
Most real lenses exhibit some amount of radial distortion, especially wide angle lenses. To obtain accurate results it is therefore important to take this effect into account. Radial distortion can approximately be modeled by having the distance from the principal point after distortion be a low-order even polynomial of the distance before distortion. κ_i are the distortion coefficients. For a typical lens one or two coefficients are sufficient to remove most of the distortion, but for wide angle lenses more coefficients might be required. There also exist specific models of distortion for fish-eye lenses and other non-standard lenses.

Planar calibration

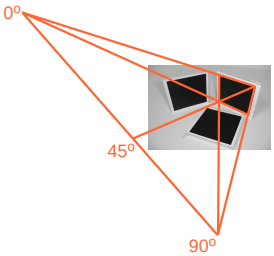
SIGGRAPH2005

- Calibration matrix relates directions to pixels

$$x = K[I \mid 0]\begin{bmatrix} d \\ 0 \end{bmatrix} \qquad d = K^{-1}x$$



- Image of square provides all directions on planar horizon



Knowing the intrinsic camera parameters allows to measure angles between incoming rays. Inversely, knowing angles between a sufficient number of incoming rays would be sufficient to compute the calibration matrix K. When imaging a square, we know the angle between two sets of parallel lines, i.e. 90 degrees. The viewing rays corresponding to those two directions –i.e. the rays corresponding to the vanishing points- therefore also have to be orthogonal. It is also simple to obtain the bisector as shown in the construction and, since 3 points form a basis for a projective line, all other angles along this line. When imaging 3 or more squares it becomes possible to establish a 2D basis in the image plane so that all angles can be determined and therefore also the calibration matrix. The planar calibration approach proposed by Zhengyou Zhang and also implemented by Jean-Yves Bouguet are based on this principle.

Planar calibration: Zhang & Bouguet

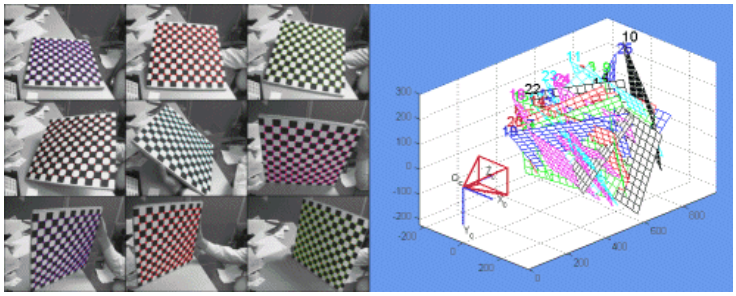
SIGGRAPH2005

- Calibration toolboxes available

http://research.microsoft.com/~zhang/calib/ [Zhang, ICCV'99/PAMI'00]

http://www.vision.caltech.edu/bouguetj/calib_doc/

(Matlab toolbox, also available in Intel's OpenCV)

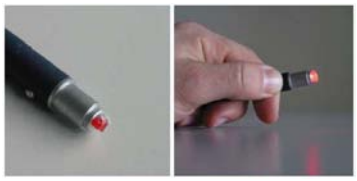


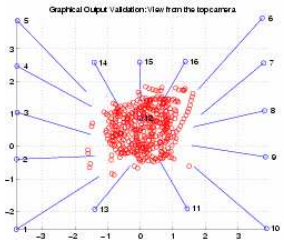
Several toolboxes have been made available that use concepts similar to the ones explained on the previous slide. A checkerboard pattern is used and the approach explained on the last slide is used to obtain an initial estimate of calibration matrix. Then, the solution is refined non-linearly taking into account radial distortion. The approach was proposed by Zhengyu Zhang at ICCV'99 and an executable is available on his webpage. Jean-Yves Bouguet has made a matlab toolbox available that implements a similar approach. Besides computing the calibration those approaches also compute the relative position of the plane with respect to the camera and vice-versa and can thus also be used to recover the relative position between multiple cameras. Bouguet's algorithm has also been implemented in Intel's OpenCV computer vision libraries.

LED-based calibration

SIGGRAPH2005

- Moving LED in front of synchronized camera system provides multi-view correspondences
- Use structure from motion





http://cmp.felk.cvut.cz/~svoboda/SelfCal/ [Svoboda et al]

http://www.cis.upenn.edu/group/tele-immersion/research/downloads/EasyCal/ [Barreto et al.]

An alternative approach that is very popular for the calibration of multi-camera rigs is to use a LED or a small light that is moved around the volume of interest. If this is done in the dark it is relatively easy to identify the LED in all the images and therefore one can easily obtain large numbers of multi-view correspondences by moving the LED around. Standard computer vision techniques can then be used to compute the intrinsic and extrinsic camera parameters for all the cameras. This approach has been implemented by many people and in many systems. Researchers have made several toolboxes available. Links to those toolboxes are given on this slide.

Camera Network Calibration

SIGGRAPH2005

- Offline Calibration Procedures
  - Planar pattern
  - moving LED
  - known points

- Requires physical access to environment
- Active Camera Networks
  - How do we maintain calibration ?

Here a brief overview of the alternative techniques for camera network calibration is given. Most of those techniques require the acquisition of specific calibration data and can therefore not be performed on-line without access to the environment and without disturbing normal operation. This is a major disadvantage in cases where it might be necessary to re-calibrate the system during operation, such as with active camera systems (e.g. with pan-tilt-zoom). A possible solution consists of using a specially prepared floor (as in the picture taken from the IBM website). In the coming slides we'll investigate a more general approach base on silhouettes.

Can we compute the calibration on-line?

SIGGRAPH2005

(data courtesy of Peter Sand)

- Feature-based?
  - Hard to match features between very different views
  - Foreground: not many features
  - Background: limited overlap
- Silhouette-based?
  - Necessary for visual-hull
  - But approach is not obvious

What are the options for on-line camera network calibration?

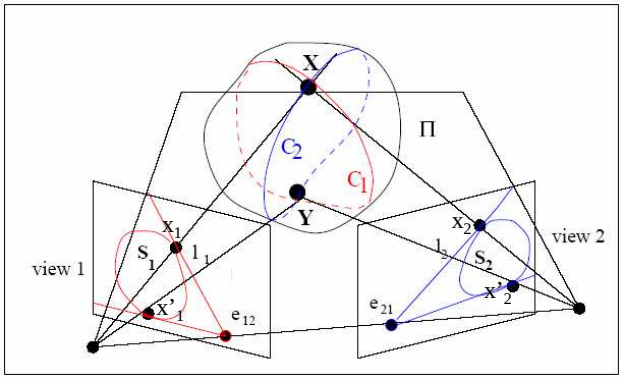
The data we use was recorded by Peter Sand at MIT for his Siggraph 2003 paper. He used a co-located motion capture system for the calibration. In the coming slides we will show how it is possible to obtain the complete calibration accurately by analysing the silhouettes. The data consists of 4 minutes of video recorded by for unsynchronized camcorders.

Multiple View Geometry of Silhouettes

- Frontier Points
- Epipolar Tangents

$$x_2^T F x_1 = 0$$

$$x_2'^T F x_1' = 0$$



- Points on Silhouettes in 2 views do not correspond in general except for projected Frontier Points
- Always at least 2 extremal frontier points per silhouette
- In general, correspondence only over two views

The key message of this slide is that there are a few specific silhouette points – the frontier points- for which the epipolar constraint has to be satisfied and that can therefore be used to compute the fundamental matrix.

Calibration from Silhouettes: prior work

Epipolar Geometry from Silhouettes

[Porril and Pollard'91],[Astrom et al'96]

Structure-and-motion from Silhouettes

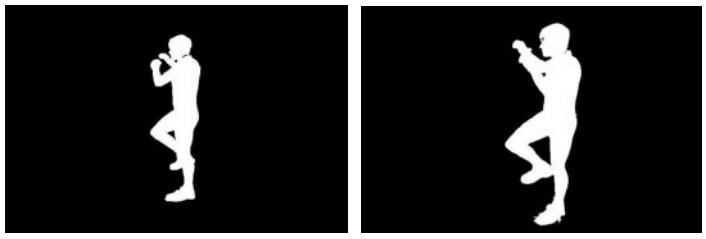
- Trinocular rig/rigid object [Joshi et al'95]
- Orthographic [Vijayakumar'96]
- Circular motion, at least to start [Wong and Cipolla'01]
- Refinement only [Yezzi and Soatto'03]

None really applicable to calibrate visual hull system

There has been a lot of prior work on calibration from silhouettes. However, surprisingly, none of those techniques allow to calibrate effectively a visual-hull system which is probably the most popular shape-from-silhouette technique.

Camera Network Calibration from Silhouettes

- 7 or more corresponding frontier points needed to compute epipolar geometry for general motion
- Hard to find on single silhouette and possibly occluded

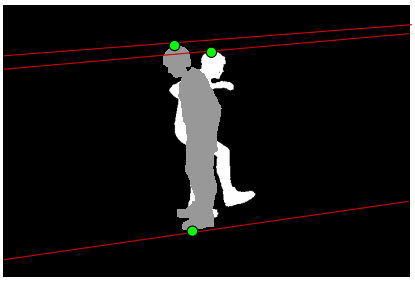


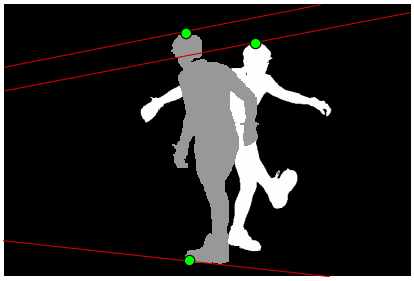
- However, Visual Hull systems record many silhouettes!

In general, to compute the epipolar geometry between two cameras, it is necessary to have 7 or more correspondences or in our case 7 or more frontier points. While it might sometimes be possible to identify those on a single silhouette, this is not true in general and many of the frontier points can be self-occluded by the object itseld (e.g. the elbow in the example given in the slide). However, in the case of a camera system observing a dynamic object, many different sets of silhouettes are available.

Camera Network Calibration from Silhouettes

- If we know the epipoles, it is simple
- Draw 3 outer epipolar tangents (from two silhouettes)





- Compute corresponding line homography H^{-T} (not unique)
- Epipolar Geometry $F=[e]_x H$

If the epipoles would somehow be known, then the rest of the epipolar geometry is easy to compute. Indeed, in this case frontier points are found as the points where epipolar lines are tangent to the silhouette. Three sets of corresponding epipolar tangents would then uniquely determine the mapping of the bundle of epipolar lines from one image to the other. Even if we only use the outer epipolar tangents (which can never be self occluded), only two pairs of silhouettes are necessary. The rest of the available silhouettes could be used to verify or refine the solution.

Let's just sample: RANSAC

[Sinha, Pollefeys and McMillan, CVPR'04]

- Repeat
  - Generate random hypothesis for epipoles ($(2\pi)^4$)
  - Compute epipolar geometry
  - Verify hypothesis and count inliers (use conservative threshold, e.g. 5 pixels, but abort early if not promising)

  until satisfying hypothesis
- Refine hypothesis
  - minimize symmetric transfer error of frontier points
  - include more inliers (use strict threshold, e.g. 1 pixels)

  Until error and inliers stable

We'll need an efficient representation
as we are likely to have to do many trials!

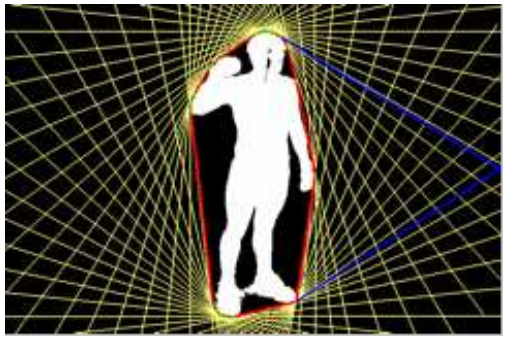
As there is no simple way to compute the location of the epipoles, we propose to just sample at random. Basically, we generate a hypothesis for an epipole by selecting two random tangents to a particular silhouette. Wherever they intersect is our epipole hypothesis. This is done in two corresponding frames (in two different cameras) independently. The size of our sampling space is bounded, i.e. (2pi)^4. Each hypothesis is verified by counting how many of the other frames support this hypothesis. Once a reasonable solution has been found it is refined using a non-linear minimization step.

A Compact Representation for Silhouettes

Tangent Envelopes

- Convex Hull of Silhouette.
- Tangency Points

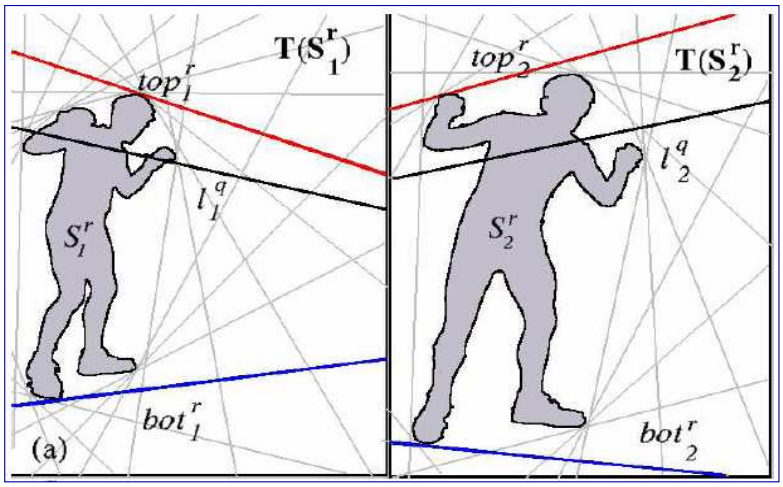
  for a discrete set of angles.



- Approx. 500 bytes/frame, i.e. small memory footprint
- Tangency Computations are efficient.

Given that we have to do this many times, it is important to have an efficient representation for our hypothesis generation and verification steps. Since we only consider the outer epipolar tangents, we can use the convex hull of the silhouettes in stead of the silhouettes themselves (which is a lot cheaper and more efficient).

Epipole Hypothesis and Computing H

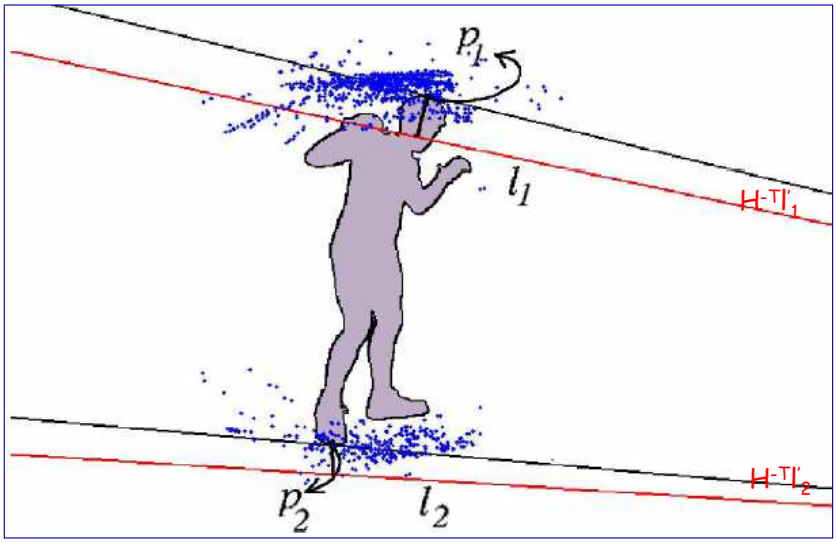


- pick two random tangents in each of two corresponding frames
- compute additional tangent pair in different frame pair

In this slide the epipolar geometry hypothesis generation is described. The additional tangent is used to uniquely determine the transfer between epipolar lines from image to the other for the hypothesis under consideration. It is important to consistently pick not only the angles at random, but also the frames that are used to be robust to incorrect silhouettes (e.g. segmentation error) and to epipoles falling within a particular silhouette.

Model Verification

SIGGRAPH2005



compute distance between tangency point $p_{1,2}$ and $H^{-T}l'_{1,2}$

In this slide the verification step of our approach is illustrated. The red line was transferred from another image and in this case is far from the actual tangent (measured in pixels at the point of tangency). Therefore, this particular frame does not support the hypothesis under consideration. The blue points show the distribution of the tangency points in the image for this hypothesis. It is typical for horizontally displaced cameras to have most points near the feet and the head.

Remarks

SIGGRAPH2005

- RANSAC allows efficient exploration of 4D parameter space (i.e. epipole pair) while being robust to imperfect silhouettes

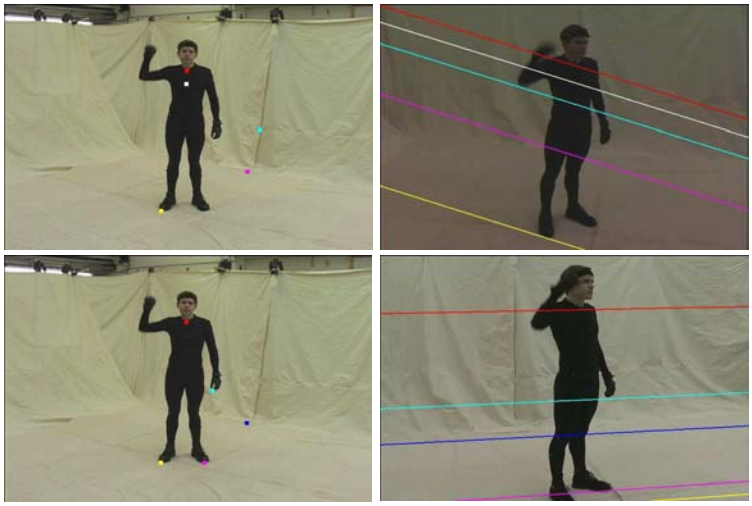
- Select key-frames to avoid having too many identical constraints (e.g. when silhouette is static for a while)

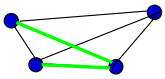
While in general RANSAC (Random Sampling Consensus) is used only for robustness, here it is also used to explore the search space for the epipoles. RANSAC efficiently handles both at the same time without having to decide how much effort is spend on robustness and how much is spend on exploration.

It is advised to discard frames that do not provide any additional information. If a person would not move for a period of time, then all those frames would provide the same information which besides being inefficient risks to confuse RANSAC (as it might find a lot of support for an incorrect hypothesis).

Computed Fundamental Matrices

SIGGRAPH2005





Typically, 1/5000 samples converges to global minima after non-linear refinement (corresponds to 15 sec. computation time)

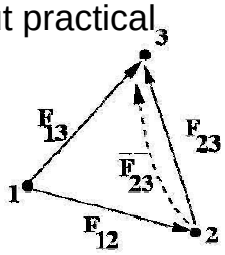
On average our approach generates one correct solution every 5000 hypothesis. This number was relatively similar for different datasets. This requires about 15 sec computation time.

From epipolar geometry to full calibration

- Not trivial because only matches between two views
- Approach similar to Levi et al. CVPR'03, but practical
- Key step is to solve for camera triplet

$$P_1 = [I|0] \qquad P_2 = [[e_{21}]_\times F_{12}|e_{21}]$$
$$P_3 = [[e_{31}]_\times F_{13}|0] + e_{31}v^T \quad (v \text{ is 4-vector})$$
$$\bar{F}_{23} = [e_{32}]_\times P_3 P_2^\dagger \quad (\text{also linear in } v)$$

Choose P_3 corresponding to $\overline{F}_{23}$ closest F_{23}

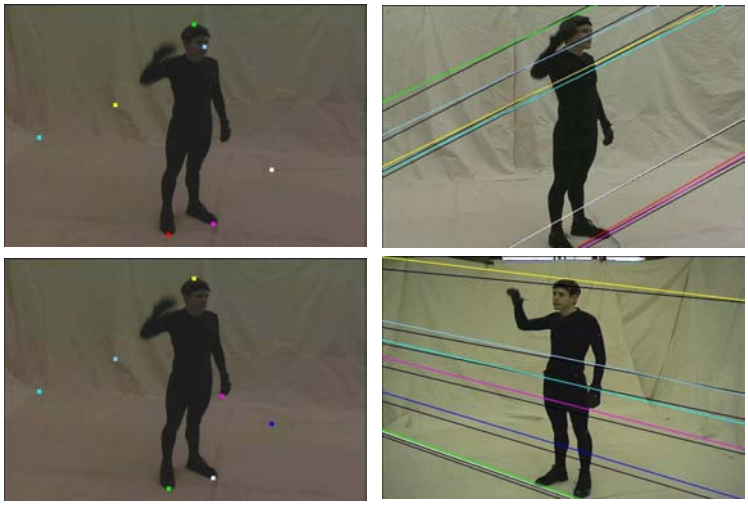


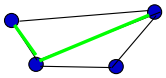
- Assemble complete projective camera network
- Projective bundle, self-calibration, metric bundle

Having computed the epipolar geometry is not sufficient. In fact, if the cameras are not yet calibrated the problem is complicated by the fact that we only have two-view correspondences (the tangency points are only visible in two views). However, given the epipolar geometries between one camera and two previously calibrated cameras, the relative pose of the new camera can also be determined uniquely in a projective frame. In fact, this step also refines one of the epipolar geometries so that it becomes consistent with the others (the result of this can be seen on the next slides). By applying this procedure recursively a complete camera network can be calibrated. Note that this does not require the epipolar geometry between all views to be computed.

Once the projective calibration is obtained, it can be refined using a bundle adjustment (for simplicity we assume that the tangency points do not move). After this self-calibration and metric bundle adjustment can be used to obtain the final calibration of the camera network.

Computed Fundamental Matrices



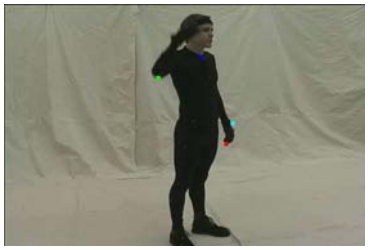


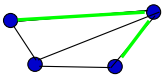
F computed directly
(black epipolar lines)

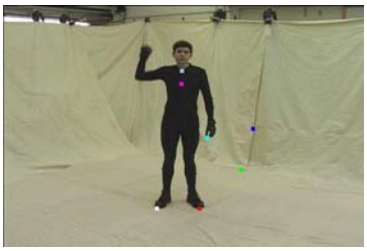
F after consistent
3D reconstruction (color)

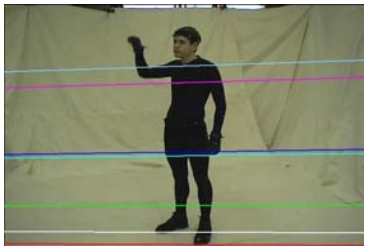
On this slide some results for computed epipolar geometries are shown. The black lines are the initial result and the colored lines are after enforcing consistency at the projective level (before projective bundle adjustment). It can be noticed that enforcing consistency already provides a significant improvement.

Computed Fundamental Matrices









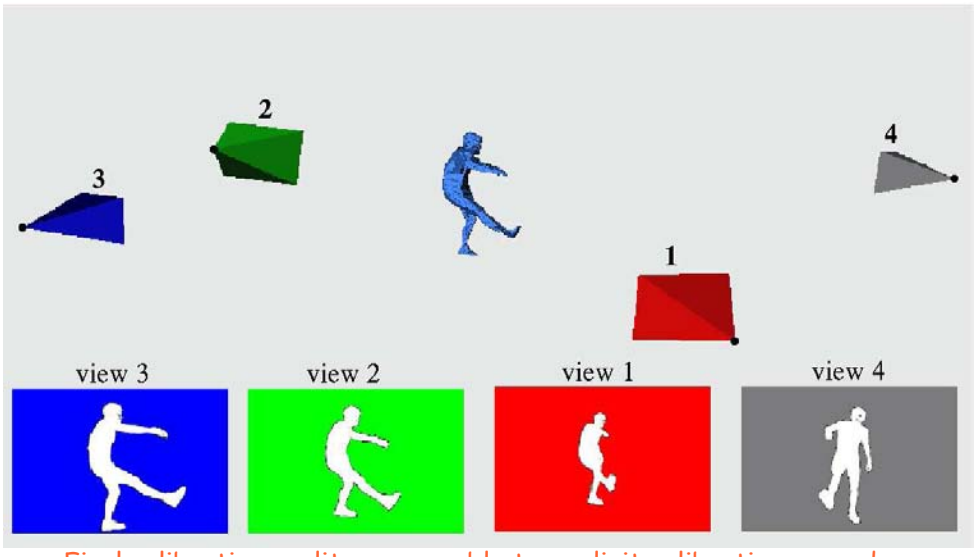
F computed directly
(black epipolar lines)

F after consistent
3D reconstruction (color)

same for two other pairs of cameras.

Metric Cameras and Visual-Hull Reconstruction from 4 views

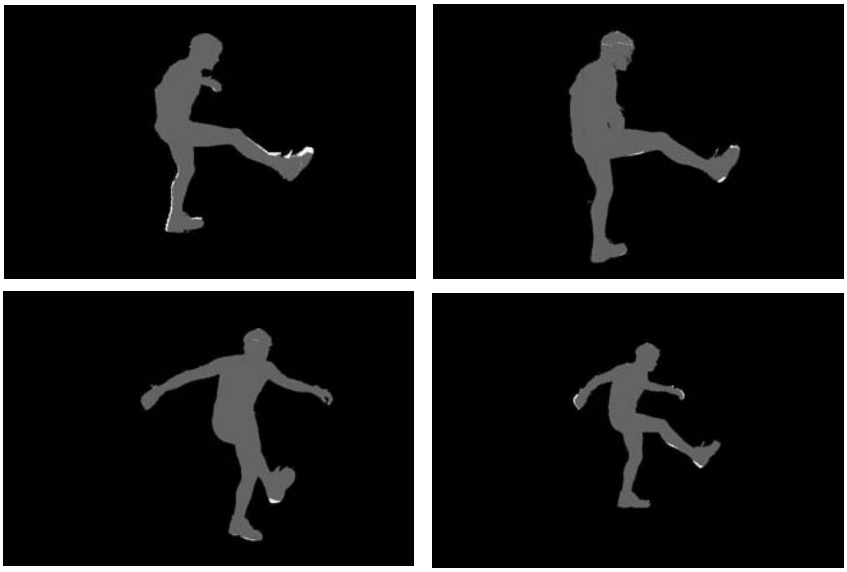
SIGGRAPH2005



Final calibration quality comparable to explicit calibration procedure

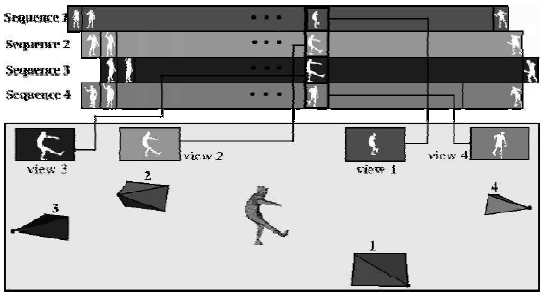
Here we show the final result of the calibration for the complete camera network and the visual-hull reconstruction for one particular set of corresponding silhouettes. Those results were obtained automatically from the four 4 minute video sequences.

Validation experiment: Reprojection of silhouettes



In this slide we verify the consistency of our results. If segmentation and calibration were perfect, back-projecting the visual hull (grey) should perfectly fill the silhouettes. The white regions are due to errors. Here it can be noticed that those are mostly located at moving body parts. Given that the cameras were unsynchronized this can be expected and is probably due to sub-frame temporal offsets between the video streams.

What if the videos are unsynchronized?



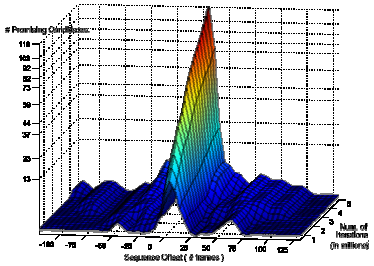
For videos recorded at a constant frame rate, same constraints are valid, up to some extra unknown temporal offsets

What can we do when the video cameras are unsynchronized so that an unknown temporal offset exists for each video stream?

Synchronization from silhouettes

[Sinha and Pollefeys, ICPR'04]

- Add a random temporal offset to RANSAC hypothesis generation, sample more ($(2\pi)^4\Delta t$)
- Use multi-resolution approach:
  - Key-frames with slow motion, rough synchronization
  - Add key-frames with faster motion, refine synchronization
  - Possibility for sub-frame synchronization

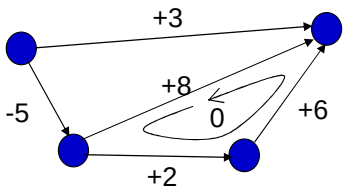


It turns out that it is feasible to sample over one more unknown. To remain efficient we use a multi-resolution approach. First we focus on slow moving silhouettes as those will allow us a rough synchronization. Then we include fast changing silhouettes to allow for a fine synchronization.

In practice, we have noticed that there is one clear peak around the correct temporal offset and that there can be a few small secondary peaks related to the repetitive nature of some of the recorded motion.

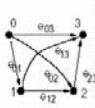
Synchronize camera network

- Consider oriented graph with offsets as branch value
- For consistency loops should add up to zero
- MLE by minimizing $\sum \sigma^{-2} \left(\Delta t - \Delta \bar{t} \right)^2$



in frames (=1/30s)

(a)



4-view network graph

e_{ij}	range	t_{ij}	σ_{ij}	$\bar{t}_{ij}$	$\hat{t}_{ij}$
e_{01}	[-13,-3]	-8.7	0.80	-8.50	-8.32
e_{02}	[-11,-1]	-8.1	1.96	-8.98	-8.60
e_{03}	[-12,-2]	-7.7	1.57	-7.89	-7.85
e_{12}	[-5,5]	-0.93	1.65	-0.48	-0.28
e_{13}	[-5,5]	0.54	0.72	0.61	0.47
e_{23}	[-6,4]	1.20	1.27	1.09	0.75

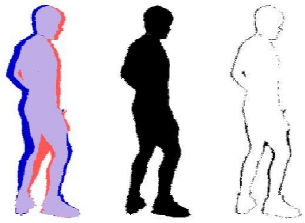
ground truth

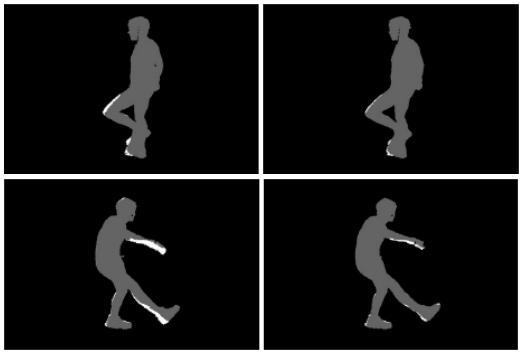
After pairwise calibration results can be refined by enforcing consistency over the whole network. The final synchronization results were accurate up to 1/3 of a frame or 1/100 of a second.

Sub-frame silhouette interpolation

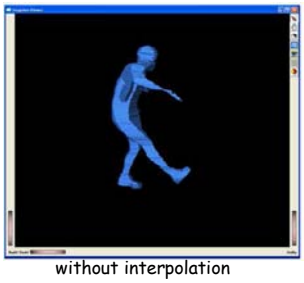
SIGGRAPH2005

[Sinha and Pollefeys, 3DPVT'04]





without interpolation with interpolation



without interpolation

with interpolation

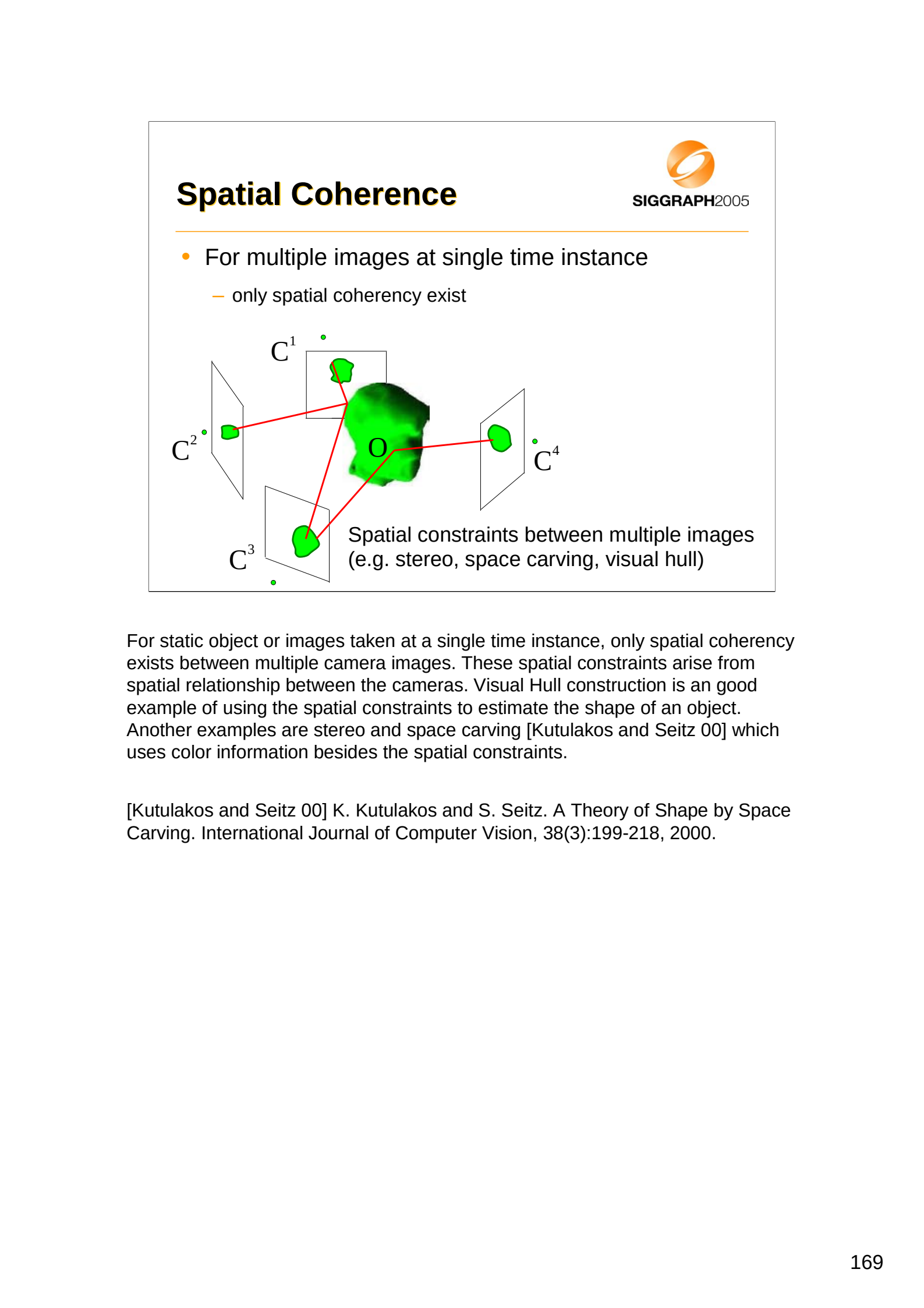
If the synchronization accuracy is sub-frame, then it makes sense to interpolate the silhouettes to predict the shape of the silhouette for exactly corresponding times. We have shown that this reduces the re-projection errors by a factor three. This is especially important for fast moving body parts.

Spacetime Coherence
11:00 – 11:30

German Cheung

Marcus Magnor

Spatial Coherence



- For multiple images at single time instance
  - only spatial coherency exist

Spatial constraints between multiple images (e.g. stereo, space carving, visual hull)

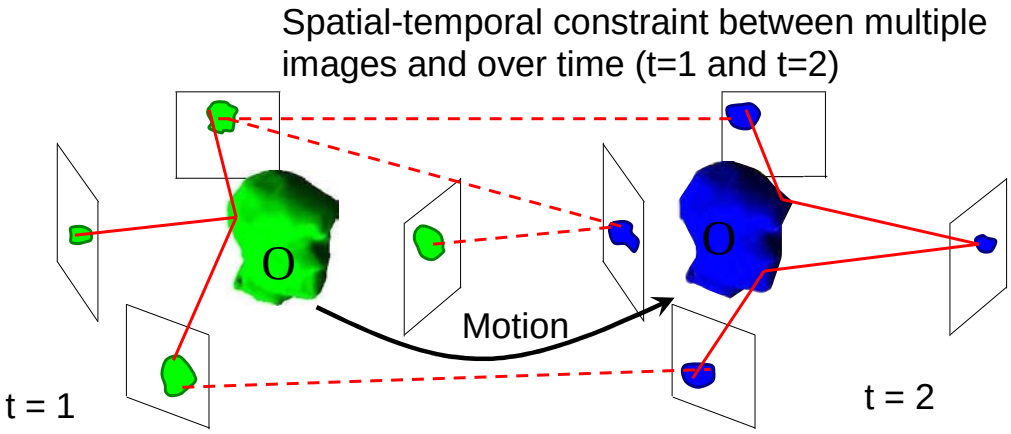
For static object or images taken at a single time instance, only spatial coherency exists between multiple camera images. These spatial constraints arise from spatial relationship between the cameras. Visual Hull construction is an good example of using the spatial constraints to estimate the shape of an object. Another examples are stereo and space carving [Kutulakos and Seitz 00] which uses color information besides the spatial constraints.

[Kutulakos and Seitz 00] K. Kutulakos and S. Seitz. A Theory of Shape by Space Carving. International Journal of Computer Vision, 38(3):199-218, 2000.

Spatial-Temporal Coherence

SIGGRAPH2005

- For multiple videos of moving objects
  - Constraints extended across time and space

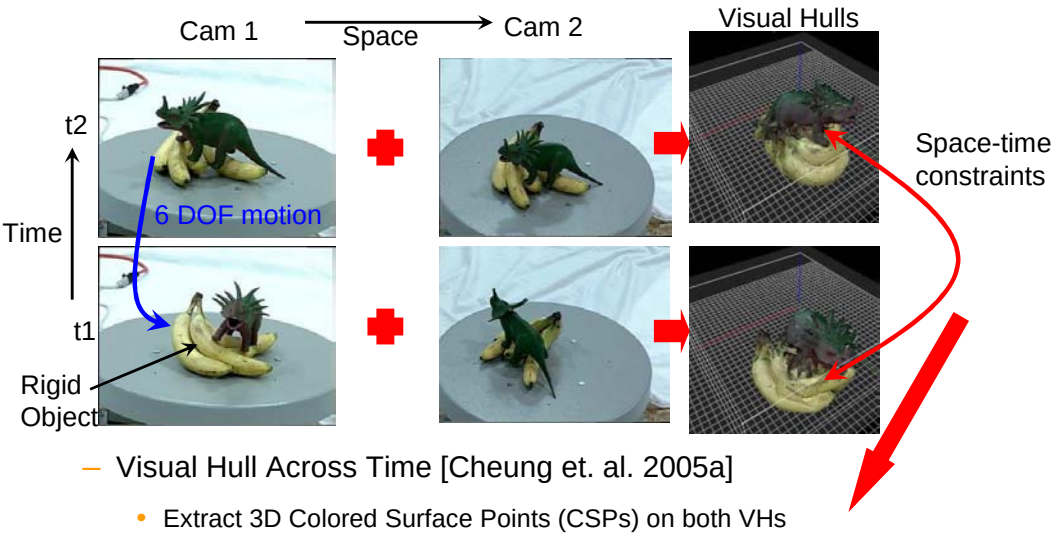


For moving (or deforming) objects in multiple video images, the constraints extend across both time and space due to the motion of the object. Generally the problem becomes more difficult because the effect of the shape and motion of the object are always coupled tightly in the video images, especially for non-rigid objects. Structure from Motion (SFM) [Tomasi and Kanade 1992] is an example which applies spatial temporal coherence to simultaneously estimate the motion and structure of a moving rigid object. Another example is space-time stereo [Zhang et. al. 2004] which can be applied to deformable objects such as faces.

[Tomasi and Kanade 1992] C. Tomasi and T. Kanade. Shape and Motion from Image Streams Under Orthography: A Factorization Method. International Journal of Computer Vision, 9(2):137-154, November 1992.

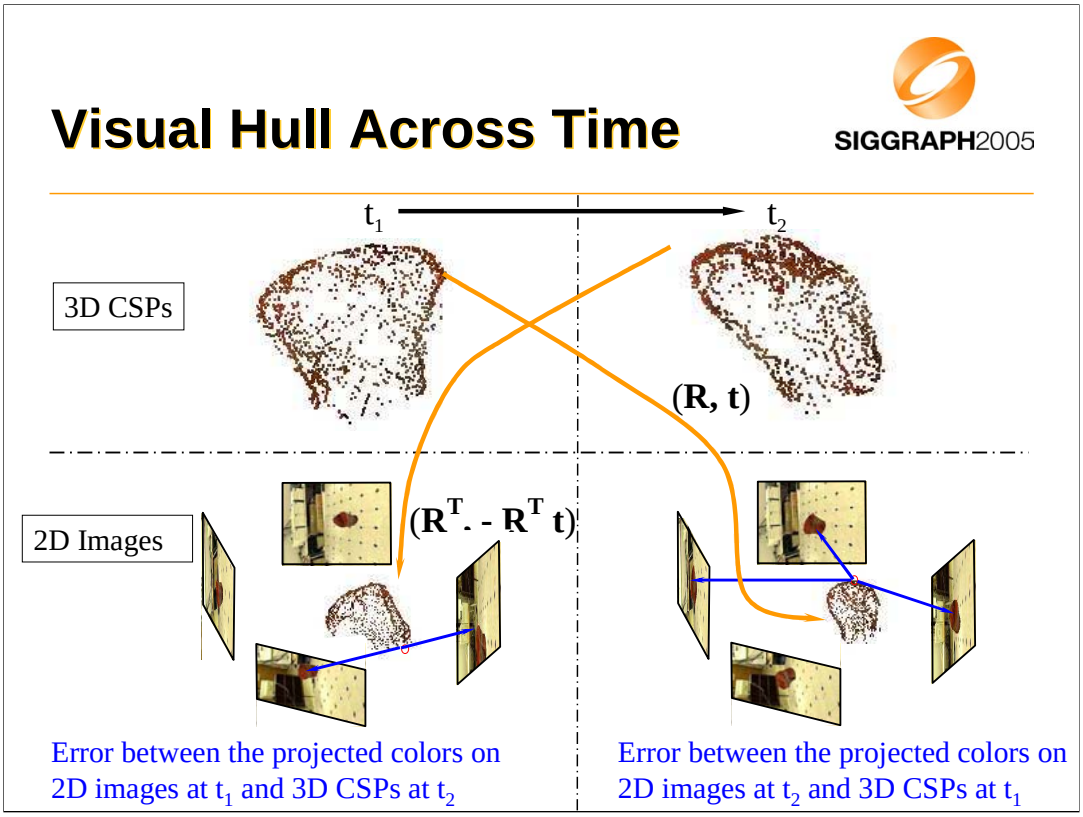
[Zhang et. al. 1004] L. Zhang, N. Snavely, B. Curless and S. Seitz. Spacetime Faces: High-Resolution Capture for Modeling and Animation, ACM Annual Conference on Computer Graphics (Siggraph 04), pages 548-558, August, 2004.

Spatial-Temporal Constraints for Visual Hull (VH)



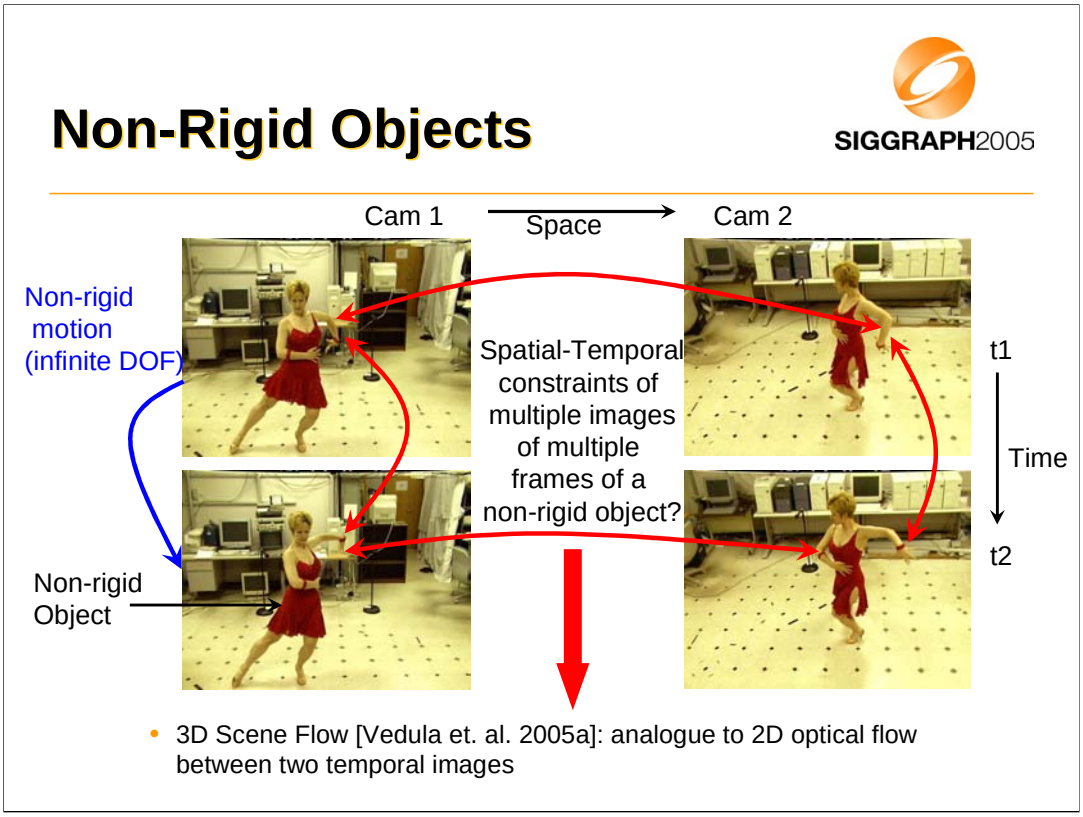
- Visual Hull Across Time [Cheung et. al. 2005a]
  - Extract 3D Colored Surface Points (CSPs) on both VHs
  - Align the CSPs across images in time and space to recover the motion

For a moving rigid object, there is only 6 Degree-of-freedom (DOF) between frames and the problem is inherently easier than deformable objects. There exists geometric constraints between the Visual Hulls of a rigid object constructed at two different frames. Combining these temporal constraints (which based on the principle of constructing VHs) with multiple-camera stereo (spatial constraints), the motion of the rigid object can be recovered using the Visual Hull Across Time algorithm: first extract points called Colored Surface Points (CSPs) on the surface of the object at each frame separately and then use these CSPs to align the Visual Hulls between two frames as shown in the next slide.



Here it shows how the spatial-temporal constraints are applied to recover the motion of an object by aligning the extracted CSPs with the silhouette and color images in an error minimization framework. The CSPs at t1 is transformed with an initial guess of the motion (of the object from t1 to t2) and the transformed CSPs are projected onto the images at t2. Two kinds of errors are used. The first is geometric error which is the distance between the projected point and the silhouette image. The second is photometric error which is the difference between the projected colors and the color of the CSPs (note that only photometric error is shown in the figure above). Similar errors are calculated by transforming the CSPs (using the inverse motion) at t2 and projecting them onto the images at t1. The 6 DOF motion can then be estimated by minimizing the combined errors. Details of the Visual Hull Across Time algorithm can be found in [Cheung et. al. 2005a].

[Cheung et. al. 2005a] K.M. Cheung, S. Baker and T. Kanade. **Shape-From-Silhouette Across Time Part I: Theory and Algorithms. International Journal of Computer Vision, Vol. 62, No. 3, May, 2005, pp. 221 - 247.**



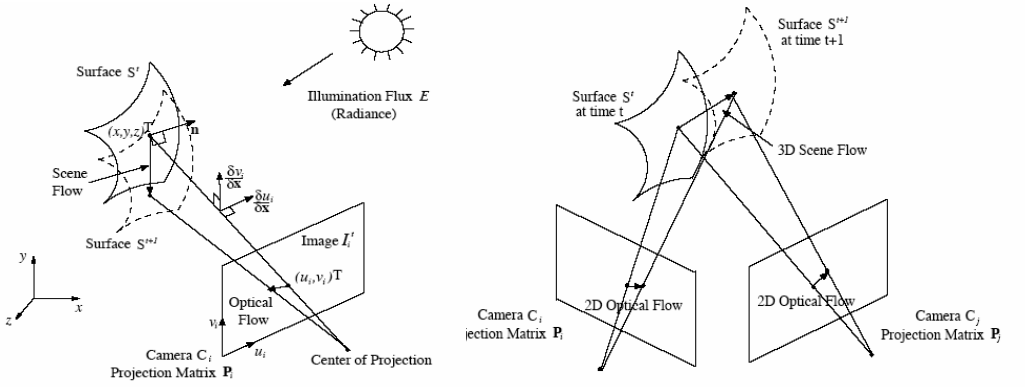
For deformable objects, the problem is much more difficult due to the infinite possible motions of each point on the object. In [Vedual et. al. 2005a], the idea of scene flow, an analogue to the 2D optical flow between two images taken successively in time, is introduced to estimate 3D motions in non-rigid dynamic scene.

[Vedual et. al. 2005a] S. Vedula, S. Baker, P. Rander, R. Collins and T. Kanade. Three-dimensional Scene Flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27, No. 3, March, 2005, pp. 475 - 480.

Scene Flow

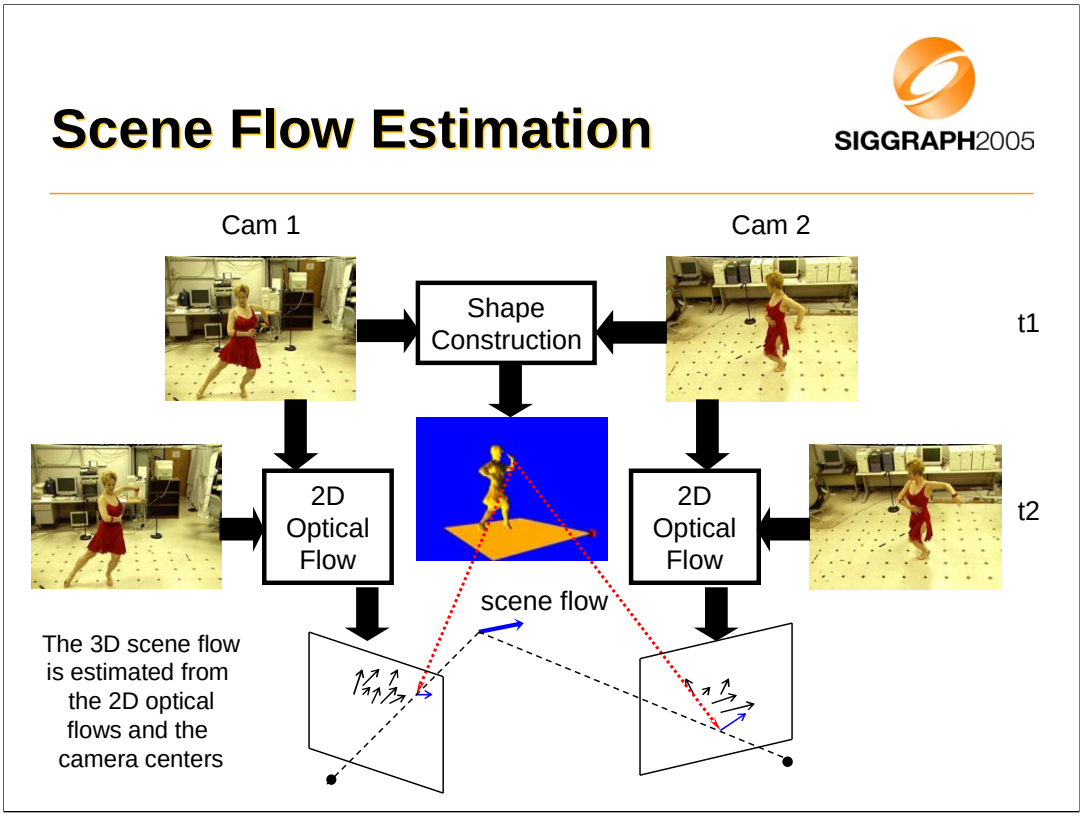
SIGGRAPH2005

- Scene flow is the (instantaneous) three-dimensional motion field of 3D points in a dynamic scene
- Optical Flow in 2D is in fact the projection of the 3D Scene Flow onto the image



Figures courtesy of Sundar Vedula

Scene flow is the instantaneous motion field of 3D points in a dynamic scene. It is especially useful in representing the motion of non-rigid object. 3D scene flow is a generalization of 2D optical flow. In fact, optical flow is the projection of scene flow on the images as shown in the figure.

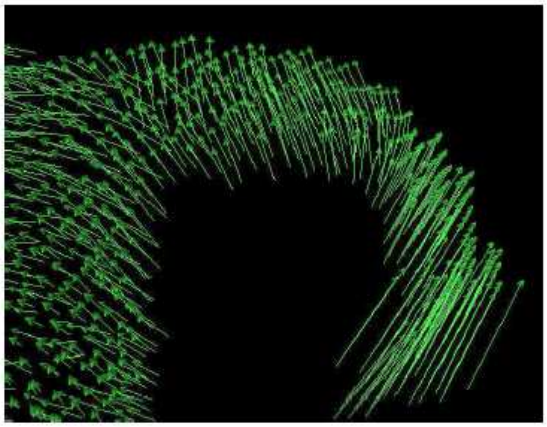


Scene flow can be estimated from a set of spatial-temporal images by a two step approach. Firstly the shape of the non-rigid object at t1 is estimated using camera images captured at t1. At the same time, 2D optical flows between t1 and t2 for each camera are estimated. To recover the scene flow of a point on the surface of the reconstructed shape, the point is projected onto the optical flows images of each camera and the 2D optical flows at the projected point are combined to estimate the 3D scene flow using the camera centers as shown in the figure. Visibility has to be taken into account when projecting the 3D surface points onto the optical flow images. In [Vedula et. al. 2001] an an algorithm is introduced to simultaneously estimate the shape and scene flow in one step while in [Carceroni and Kutulakos 2001], an algorithm is proposed to estimate scene flow in the 3D scene domain directly (instead of through 2D optical flows).

[Vedula et. al. 2001] S. Vedula, S. Baker, S. Seitz and T. Kanade. Shape and Motion Carving in 6D. In P*roceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '00)*, June, 2000.

[Carceroni and Kutulakos 2001] R. Carceroni and K.Kutulakos. Multi-view Scene Capture by Surfel Sampling: From Video Streams to Non-rigid 3D Motion, Shape and Reflectance. In Proceedings of the IEEE International Conference on Computer Vision, pages 60-67, 2001.

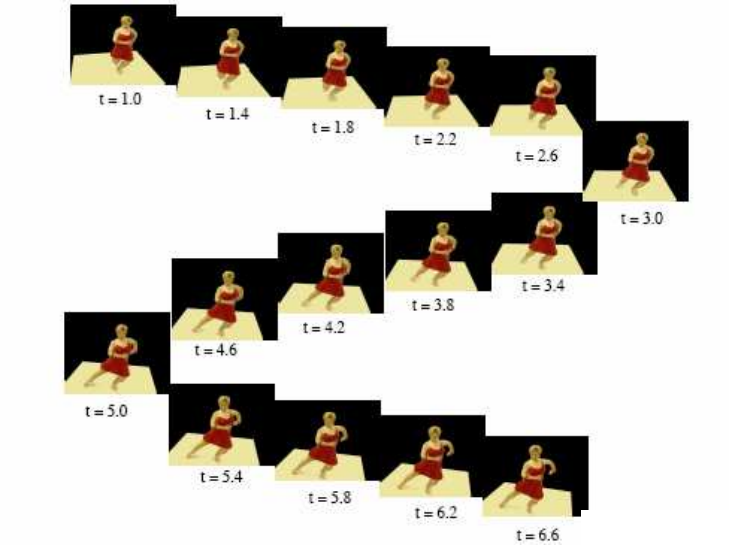
Examples of Recovered Scene Flows



Images courtesy of Sundar Vedula

Here are examples of the recovered scene flow of the dance example shown in the previous slide.

Application: Spatial Temporal View Interpolation

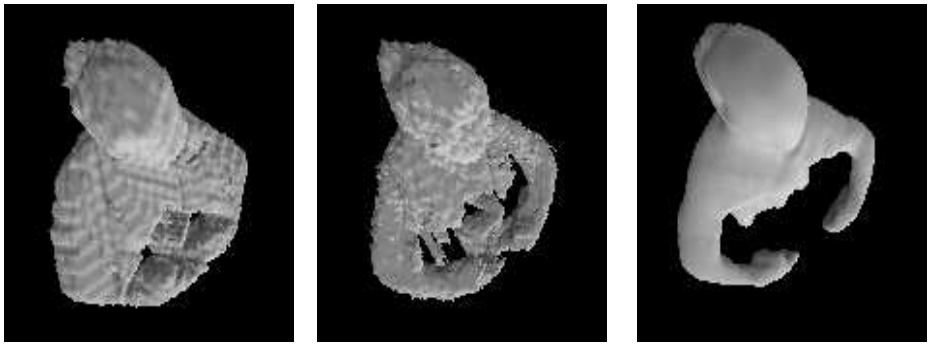


Images courtesy of Sundar Vedula

One application of scene flow is for interpolating a dynamic scene both spatially and temporally. The scene flows provide a continuous motion flow of the deformable object which can be used to change the speed of the object during rendering to create special effects such as slow motion or motion blur. Details of this application can be found in [Vedula et. al. 2005b].

[Vedula et. al. 2005b] S. Vedula, S. Baker and T. Kanade. Image-Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events. ACM Transactions on Graphics, April, 2005.

Spacetime-coherent Reconstrution



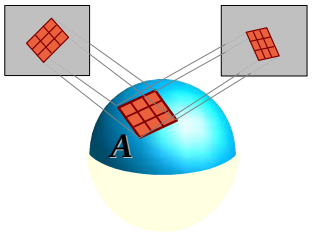
Per-time step Visual Hull | Per-time step Photo Hull | Spacetime-coherent surface

Having video streams available allows us to exploit the fact that motion in nature is continuous (we exclude quantum leaps). By making global use of temporal coherence, we can obtain much better reconstruction results than if we reconstruct geometry from each time frame separately.

In the example shown, eight equally-space video cameras recorded a dancer from the perimeter of the stage. All cameras where positions at about eye level, no camera recorded the dancer from the top. At the depicted time instant, the dancer holds her arms in front of her torso, so no camera has a clear view of the area between arms and the torso. If reconstructed only from the images of this moment, the visual hull encloses the volume between arms and torso, and also the photo hull cannot do much better. By enforcing temporal coherence over all time frames, however, the temporally obstructed region gets automatically interpolated to be consistent with previous and later time steps, leading to a much more plausibly reconstruction result.

Spacetime-coherent Reconstruction

- 3D surface in 4D spacetime
  - Weighted minimal surface $A(\Sigma) = \int_{\Sigma} \Phi(s, \mathbf{n}(s))\, dA(s)$
  - Photo-consistency $\Phi(s,\mathbf{n}) := -\frac{1}{V_{s,t}} \sum_{i,j=1}^{l} \upsilon_i^t(s)\ \upsilon_j^t(s) \frac{\chi_{i,j}^t(s,\mathbf{n})}{A(\Diamond_{s,\mathbf{n}})}$
    - cameras i,j
    - visibility υ_i
    - No. camera pairs $V_{s,t}$
    - Normalized dross correlation $\chi_{i,j}$
    - Cross correlation area A



To exploit temporal coherence, we regard the time-varying object surface as a three-dimensional hyper-surface in four-dimensional spacetime. If we cut this 3D hyper-surface along a hyper-plane of constant time, we get the 2D object surface in 3D space at that time instant.

We first write down our problem in theoretical terms. We want to find the 3D hyper-surface \Sigma that, when integrating an error functional \Phi over its domain, yields a minimum error. To quantify the error, we use the photo-consistency of all camera images.

The same approach has been pursued to solve different problems in computer vision, e.g. active contours (snakes), tracking, as well as static surface modeling.

Weighted Minimal Hyper-Surfaces

- Wanted: k-dimensional hyper-surface $\Sigma \subset R^{k+1}$
- that minimizes $A(\Sigma) = \int_{\Sigma} \Phi(s, \mathbf{n}(s))\, dA(s)$
  - Weighting fct. can depend on position and normal
- Necessary condition

$$\langle \Phi_s, \mathbf{n} \rangle - \mathrm{Tr}(\mathbf{S})\Phi + \mathrm{div}_{\Sigma}(\Phi_{\mathbf{n}}) = 0$$

Euler-Lagrange equation

[Goldluecke & Magnor, ECCV'04]

So far, we have only formulated the problem in mathematical notation. To explicitly state the hyper-surface \Sigma that minimizes the integral over the weight function \Phi, we have to come up with a different equation that can be solved.

As it turns out, the necessary condition for \Sigma to minimize the error integral can be stated as a Euler-Lagrange equation. In fact, this partial differential equation is valid for arbitrary dimensions k and weight functions \Phi that depend on position s and normal direction n(s).

•\Phi_s and \Phi_n denote the derivative of \Phi with respect to s and n

•< , > denotes the dot product

•S is the second fundamental tensor, also referred to as shape operator or Weingarten map, and Tr(S) = div(-n)

Numerical Solution

- Evolution equation
  - τ : evolution parameter
  - Level-set method
    - $u(x,\tau) > 0$ outside Σ_τ
    - $u(x,\tau) < 0$ inside Σ_τ
    - $u(x,\tau) = 0$: surface
- Differential operator
  - Identities
  - Product rule in reverse

$$\frac{\partial}{\partial \tau}\Sigma_\tau = \Psi \mathbf{n}$$

$$u(x,\tau) = 0 \Leftrightarrow x \in \Sigma_\tau$$

$$\mathbf{n} = -\frac{\nabla u}{|\nabla u|} \Rightarrow |\nabla u| = -\langle \nabla u, \mathbf{n} \rangle$$

$$\frac{\partial}{\partial \tau} u\Big|_{\Sigma_\tau} = -\left\langle \nabla u, \frac{\partial}{\partial \tau}\Sigma_\tau \right\rangle = -\langle \nabla u, \mathbf{n} \rangle \Psi = \Psi |\nabla u|$$

$$\Psi = \langle \Phi_s, \mathbf{n} \rangle - \mathrm{Tr}(\mathbf{S})\Phi + \mathrm{div}_\Sigma(\Phi_\mathbf{n})$$

$$\mathrm{Tr}(\mathbf{S}) = \mathrm{div}\left(\frac{\nabla u}{|\nabla u|}\right)$$

$$\mathbf{n} = -\frac{\nabla u}{|\nabla u|}$$

$$\Psi = \left[-\mathrm{div}\left(\Phi \cdot \frac{\nabla u}{|\nabla u|} \right) + \mathrm{div}_\Sigma(\Phi_\mathbf{n}) \right]$$

To find the solution to the Euler-Lagrange equation, we make use of an iterative approach. Starting from an initial surface (to which we'll get in a second), the evolution equation updates the surface \Sigma_\tau by moving all surface points in the (negative) normal direction by an amount set by the differential operator \Psi. The iteration terminates when the stationary solution of the evolution equation is found, i.e., when \Psi=0 everywhere on the surface.

We use the level-set method to numerically evolve the surface, i.e., we the update grid cells u for each iteration step tau. The surface \Sigma corresponds to u=0, the zero level set. It is implicitly defined in-between neighboring grid cells where u becomes negative (or positive).

For each iteration step, the differential operator \Psi must be numerically evaluated from local grid point values u. Using two identities and applying the product rule of calculus in reverse, we can derive a representation of \Psi that can be numerically computed.

Practical Implementation

- Evolution equation

$$\left.\frac{\partial}{\partial \tau} u\right|_{\Sigma_\tau} = \Psi |\nabla u|$$

- Differential operator

$$\Psi = \left[-\operatorname{div}\left(\Phi \cdot \frac{\nabla u}{|\nabla u|} \right) + \operatorname{div}_\Sigma (\Phi_{\mathbf{n}}) \right]$$

- Iterative solution

$$u_{i+1}^{xyzt} = u_i^{xyzt} + \Psi\left(u_i^{xyzt}\right) \left| \nabla u_i^{xyzt} \right| \cdot \Delta\tau$$

  - Narrow-band
  - Adaptive step size

$$\Delta\tau \le \max_{(x,y,z,t)} \left(\frac{\operatorname{diam\,cell}(x,y,z,t)}{\left| \Psi\left(u_i^{xyzt}\right) \cdot \nabla u_i^{xyzt} \right|} \right)$$

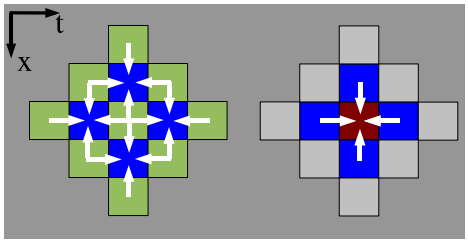
    - CFL condition

We evaluate the evolution equation on a regular 4D grid u^xyzt (3 spatial + 1 temporal dimensions). During each iteration step, grid cell values are updated. To reduce computation, we do not compute values for all grid cells but only for cells that are close the zero level set. Step size \Delta\tau must be adapted for each iteration step to meet the Courant-Friedrichs-Levy condition that ensures that the zero-level set surface does not wander by more than maximally one grid cell diameter (diam cell) per iteration step.

Differential Operator – Numerical Evaluation

SIGGRAPH2005

$$\Psi = \left[-\mathrm{div}\left(\Phi \cdot \frac{\nabla u}{|\nabla u|} \right) + \mathrm{div}_{\Sigma}(\Phi_{\mathbf{n}}) \right]$$



– Finite differences

1. ***n*** blue cell
2. Φ blue cell
3. *div(Φn)* red cell
4. Φ_n blue cell
5. $div_{\Sigma}(\Phi_n)$ red cell

$$\mathrm{div}(\Phi\mathbf{n})\big|_{\mathbf{p}=(x,y,z,t)} = \sum_{i=1}^{4} \frac{\Phi^{\mathbf{p}+\mathbf{e}_i} n_i^{\mathbf{p}+\mathbf{e}_i} - \Phi^{\mathbf{p}-\mathbf{e}_i} n_i^{\mathbf{p}-\mathbf{e}_i}}{2}$$

The differential operator must be evaluated for each grid cell individually. Because of its second-order derivatives, we need values from grid cells up to 2 cells away from the grid cell under consideration. In the image on the top right, we have collapsed the three spatial dimensional into one dimension. The center cell has actually 8 directly neighboring (blue) grid cells (6 spatial plus 2 temporal direct neighbors).

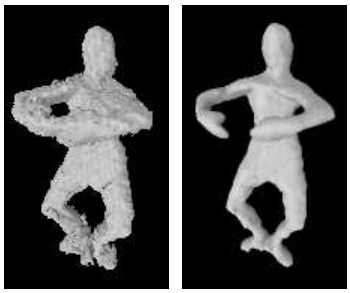
1. Using finite differences, we first compute the 4D normal directions of all adjacent (blue) cells from the values of the neighboring (green) grid cells.
2. The error functional \Phi is determined for the blue cells.
3. From the blue cell values, div(\Phi n) is computed for the center (red) cell as shown on the lower right.
4. The second term is more complicated to evaluate. First \Phi_n is computed for the blue cells using finite differences.
5. From \Phi_n at the blue cells, a 4x4 matrix U=\Phi_ns is set up for the center (red) cell. From an arbitrary orthonormal base of the tangent plane in s, the 3x3 matrix V_ij=b^T_i U b_j is computed. The trace of V, Tr(V) is then equal to the second term of the differential operator (for details see [Goldluecke & Magnor, CVPR'04])

First, the normal directions of blue cells' normals are

characteristics

To numerically evalutate differential

Grid Initialization



- Visual Hull
  - +1 inside
  - -1 outside
  - -1..1 on the boundary
- Convergence towards minimum
  - Evolution equation
    - “Force” against normal direction

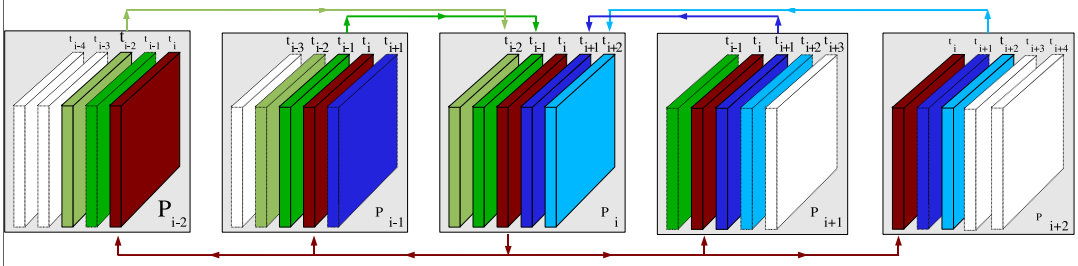
The grid cells must be initialized to start iterating the evolution equation. The visual hull turns out to be very well suited for grid initialization. From the segmented input images, we can easily determine the visual hull for each time step as a voxel model. All voxels (grid cells) inside the visual hull are set to +1, while all grid cells outside are set to -1. Boundary voxels can be assigned an intermediate value derived from partial silhouette projection.

Note that the Euler-Langrange equation that we just solved is a necessary condition, i.e., it can in theory also yield a maximum value of the energy function. However, the visual hull has the advantageous property that it is always at least as large or larger than actual object geometry. By always updating the surface against surface normal direction, i.e., inward, we can be sure that our evolution equation always converges towards a local minimum.

In theory, we are not guaranteed to find the global minimum. We found, however, that the convergence results appear very plausible. Intuitively, the large number of time steps constraints the space of likely solutions so much that, when starting from the visual hulls, the closest minimum appears to be at least very close to the global minimum.

Distributed Computation

SIGGRAPH2005



- Process P_i :
  - Transmits time slice $\sum_t t_i$
  - Receives time slices $\sum_t t_{i\pm1,i\pm2}$
  - Evaluates differential operator Ψ for cells near zero level set
  - Transmits maximum value to determine new step size

As can be guessed, the algorithm is computationally expensive. For many cells of the four-dimensional grid, the differential operator must be repeatedly evaluated. Because for operator evaluation, each grid cell requires values only from nearby cells, the algorithm can be parallelized to run on a distributed system. For example, the algorithm can be subdivided into multiple processes where each process reconstructs the spatial grid cell values at one time step t (red slice). On a cluster, for example, each process runs on a separate processor, and the 3D grid for the time step is kept in local memory. For its computations, the process (red) requires grid cell values from the 2 previous and the next 2 time steps, reconstructed by other processes. On the other hand, the process also sends the reconstruction for time step t to its neighbor time-step processes.

To determine the step size for the next iteration step from the CFL-criterion, the maximum operator value of each process is sent to a separate process. The algorithm stops when the maximum operator value falls below a preset threshold, indicating that a stationary solution of the surface evolution equation has been reached.

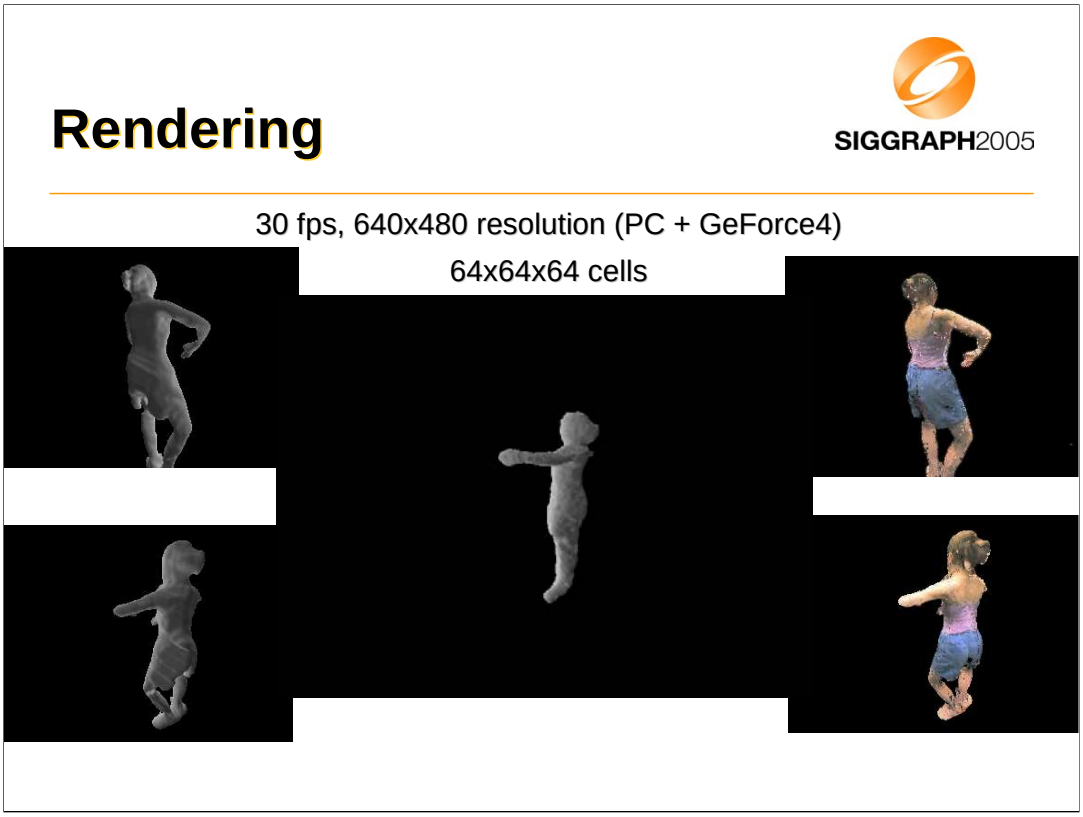
Computational Cost

- Grid resolution
  - Up to 128x128x128 cells per time slice
  - As many time slices (=processes) as time frames
- Computation

grid res.	32^3	64^3	128^3
time [s]	60	360	1200
Memory	80MB	180MB	535MB

  - Sun Fire 15K, 75 UltraSPARC III+ 900MHz

Here are some figures to evaluate computational cost. Because temporal coherence is taken into account globally, all images and grid cells must be accessible simultaneously. To avoid swapping data from and to hard drive, we use a small computer cluster. On each processor node, one or more processes are run, while the associated reference images and reconstructed grid cell values are stored in local memory.



To render the zero level set model, we can either reconstruct triangle meshes using, e.g. marching cubes. Alternatively, we render the grid cells adjacent to the zero level set using billboard rendering [Goldluecke & Magnor, ICIP'03]. At 64^3-voxel resolution, the dynamic model can be updated and rendered at real-time frame rates on conventional graphics hardware.

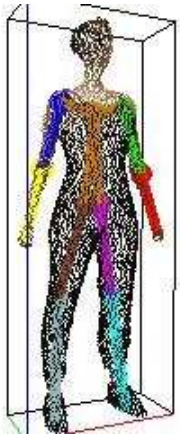
Model-based VBR
11:30 – 12:00

German Cheung

Christian Theobalt

Human Kinematics Modeling

- Advantages:
  - Video-based: non-invasive
  - Automatic: no manual intervention
  - 3D shape+skeleton+segmentation+texture
  - Built from scratch, no generic models or primitives
  - Single system: modeling+marker-less motion capture



Although there exists precise full body laser-range scanner to capture human body shape, they are usually expensive and difficult to use. Moreover, they do not recover the very important joint skeleton information of the body. A vision-based human kinematic modeling system has the advantages of being non-invasive and relatively cheaper. In particular the system to be described uses multiple video streams to recover the 3D shape, skeleton, segmentation and texture information of a human body automatically without human intervention and the need of pre-existing models (the only knowledge is the joint structure of the human body). The same camera system set up can also be used for marker-less motion capture once the kinematic model of the person is built.

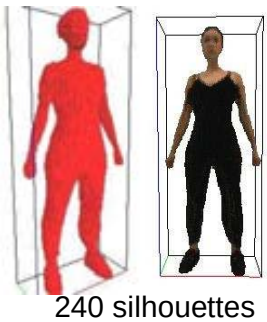
Basic Idea: Refined Shape (Visual Hull) Over Time

SIGGRAPH2005

8 silhouettes
from one time instance

QuickTime™ and a
YUV420 codec decompressor
are needed to see this picture.

Silhouette sequences

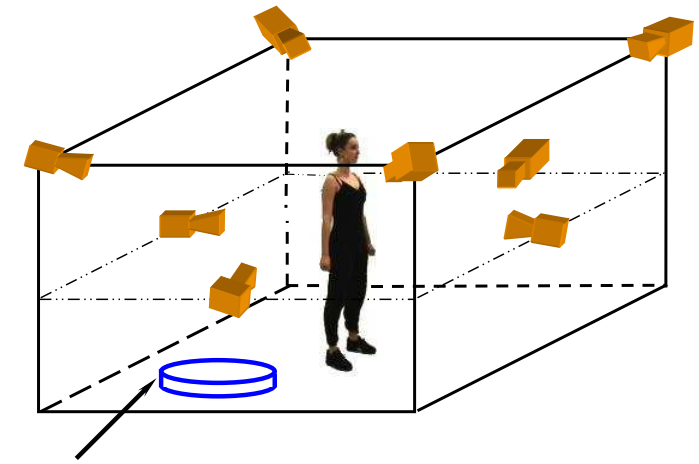


240 silhouettes
from 30 time instances

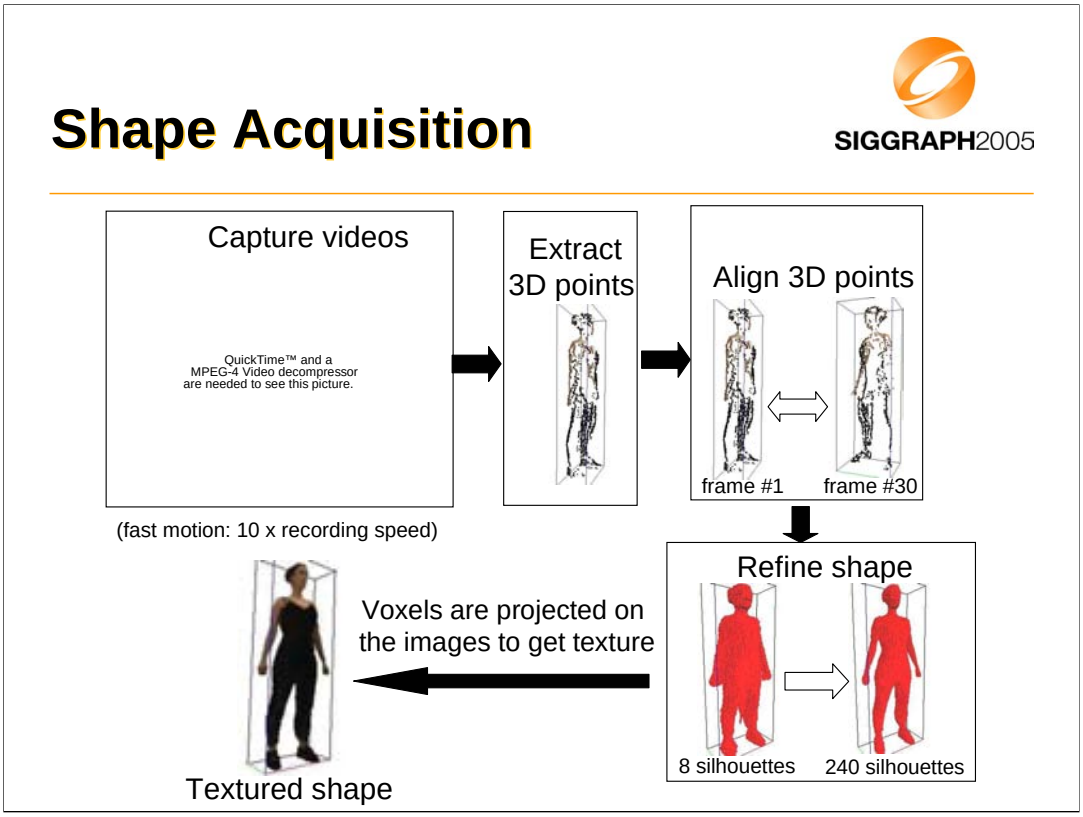
- Body is rigid
- Motions (need not to be known) between frames

The main technical idea behind the vision-based modeling system is Visual Hull Across Time as discussed previously. Visual Hull is a great method to estimate the shape of an object. However, VH constructed using small number of cameras are very coarse and therefore not useful in body shape modeling. The shape estimate can be improved drastically (as shown in the images above) by combining and refining VHs constructed over time. The basic assumptions made are that the body is rigid (which can be made valid as shown in the next slide) with rigid motions (the actual motions do not have to be known as they will be estimated in the process) from the captured video.

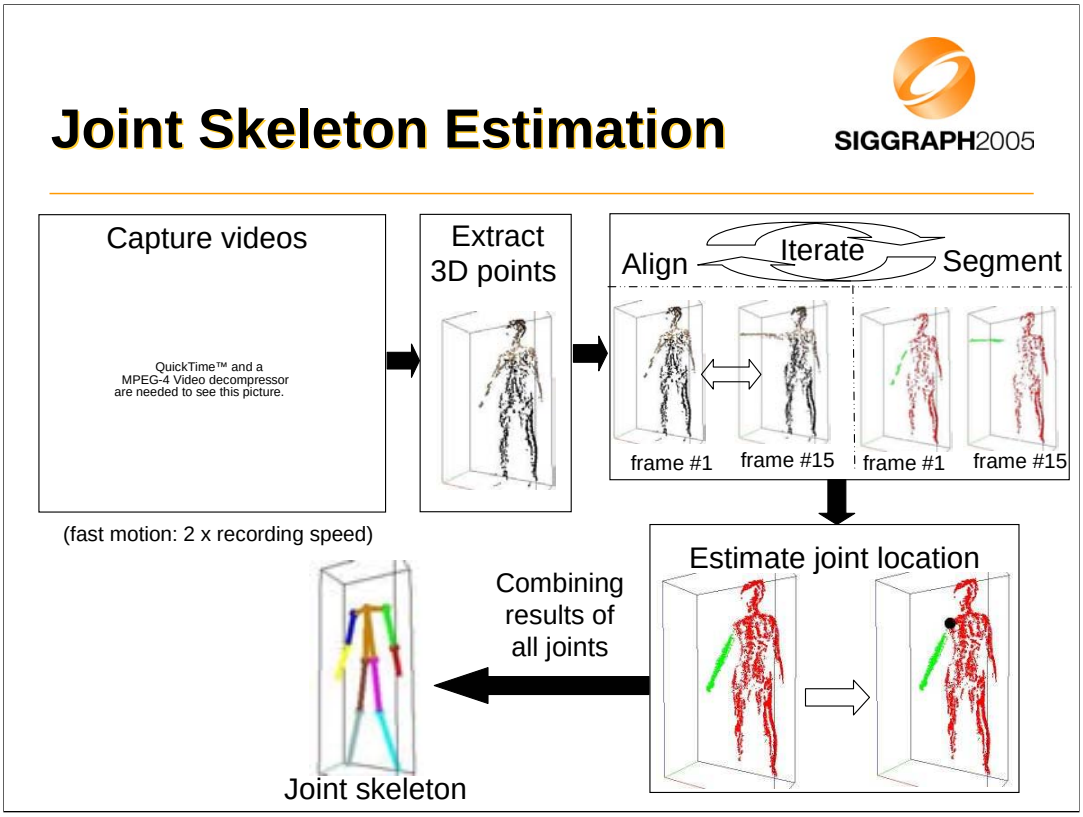
System Setup



The system setup is fairly simple with 8 synchronized, calibrated and color balanced cameras. To facilitate the capturing process, a un-calibrated turn-table with medium speed (2 rev./min.) is used to induce motions of the whole body.

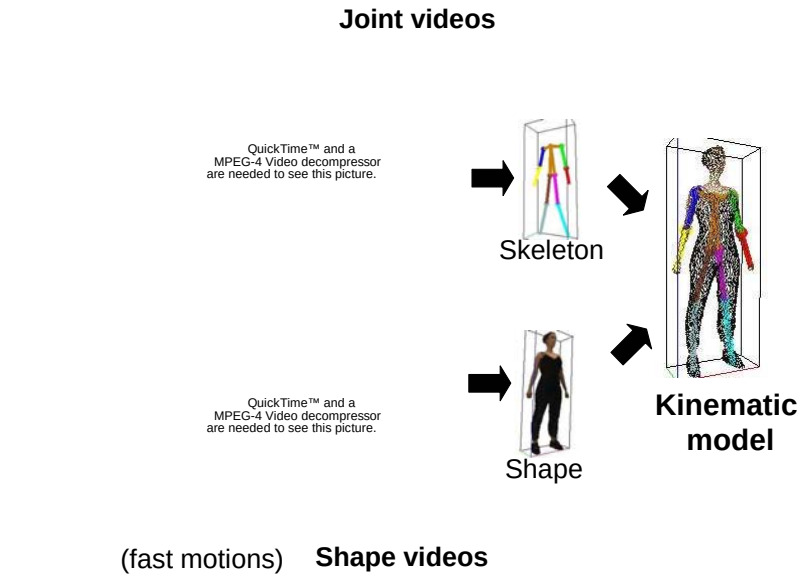


The subject is asked to remain still (for the rigidity assumption) on the turntable for 30s during capture. By extracting the 3D CSPs (please refer to the notes in the space-time section) and align them over time, the motion of the body in the video is estimated. Once the motions have been recovered, the silhouette images extracted over the whole sequence are used to build a precise shape model of the body. The voxels are projected onto the color images to get the texture of the body shape. Note that space carving can be used instead of VH construction once the motions are estimated.



By using a similar idea, the joint locations of the body are estimated by asking the subject to exercise each joint one at a time so that the body can be considered as a two-link articulated object. The VHs are iteratively aligned and segmented into the two (linked) body parts. Once the motion and segmentation of all frames have been estimated, the joint location can be estimated easily by solving a least square problem.

Merging Shape & Skeleton

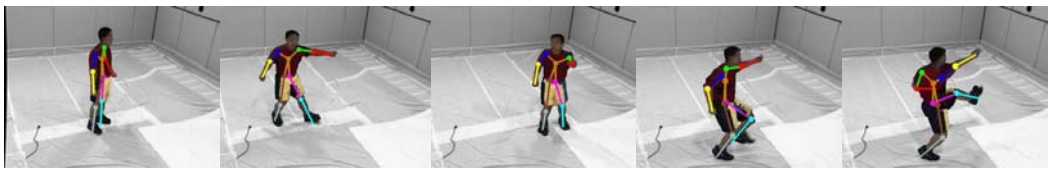


Once the body shape and joint skeleton have been estimated, they are merged together to form a complete kinematic model of the subject.

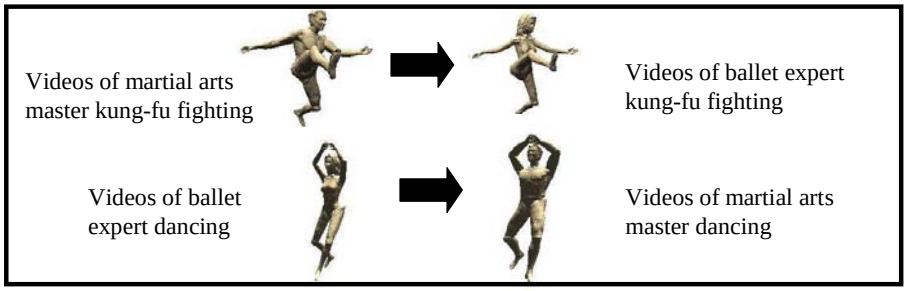
Application: Video-based Motion Transfer

SIGGRAPH2005

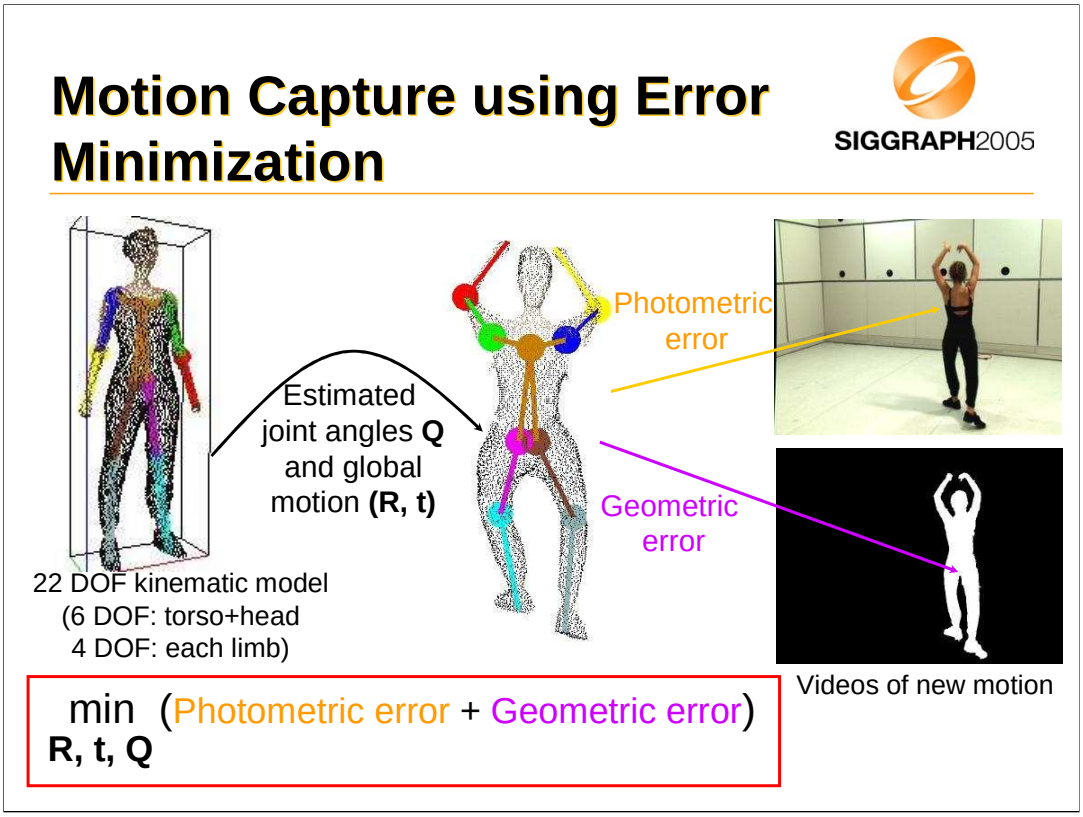
- Marker-less motion capture



- Video-based motion transfer



Once the kinematic models have been built, they can be used to perform marker-less motion capture and video-based motion transfer. The same body modeling camera system is used to track the motions of the subjects. Photo-realistic videos of one subject performing the motion of the other is rendered using a video-based rendering algorithm.

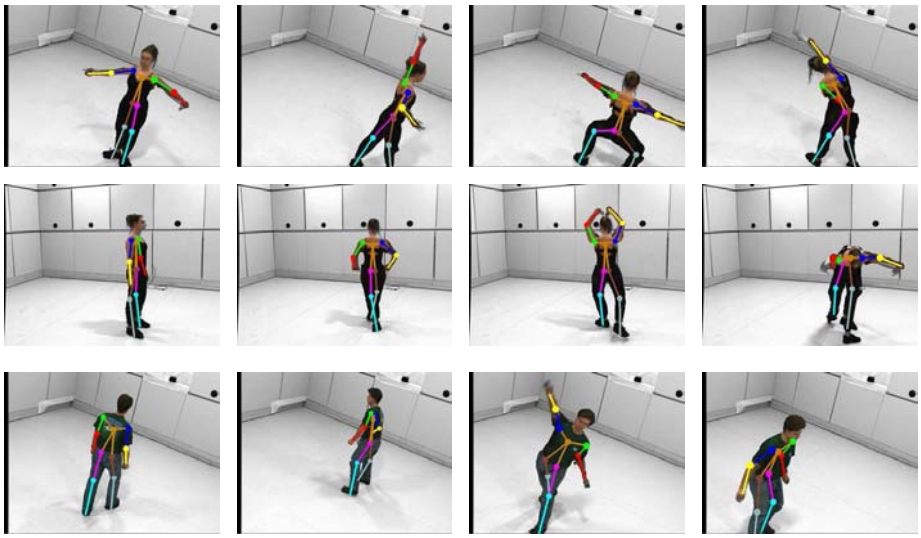


Motion tracking can be formulated as a minimization problem. With an estimate of the motion parameters (the joint angles), the model is projected back onto the color and silhouette images. The photometric and geometric errors between the projected model and the images are used as the minimization criteria. Details of both the kinematic modeling and marker-less motion capture system and algorithms can be found in [Cheung et. al. 2005b].

[Cheung et. al. 2005b] K.M. Cheung, S. Baker and T. Kanade. **Shape-From-Silhouette Across Time: Part II: Applications to Human Modeling and Markerless Motion Tracking.** *International Journal of Computer Vision*, Vol. 63, No. 3, August, 2005, pp. 225 - 245.

Marker-less Motion Capture

SIGGRAPH2005

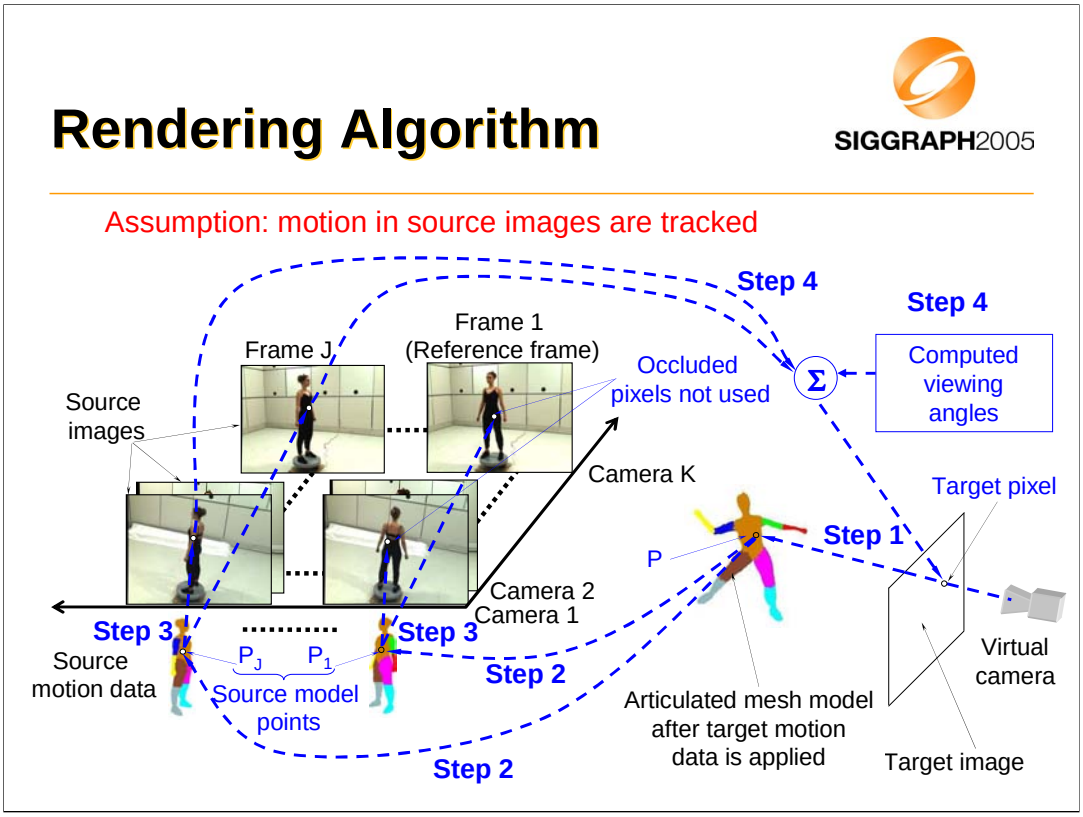


Here are some example frames from a tracked dancing sequence and a tracked throwing sequence from two different subjects.

Video-based Motion Transfer

- Differences from other IBR algorithms
  - Moving articulated object (human bodies)
  - “Extrapolation” (new motion to different person) vs. “Interpolations” (replay motion of same person)
  - Pixels come from an ensemble of images across both time and space

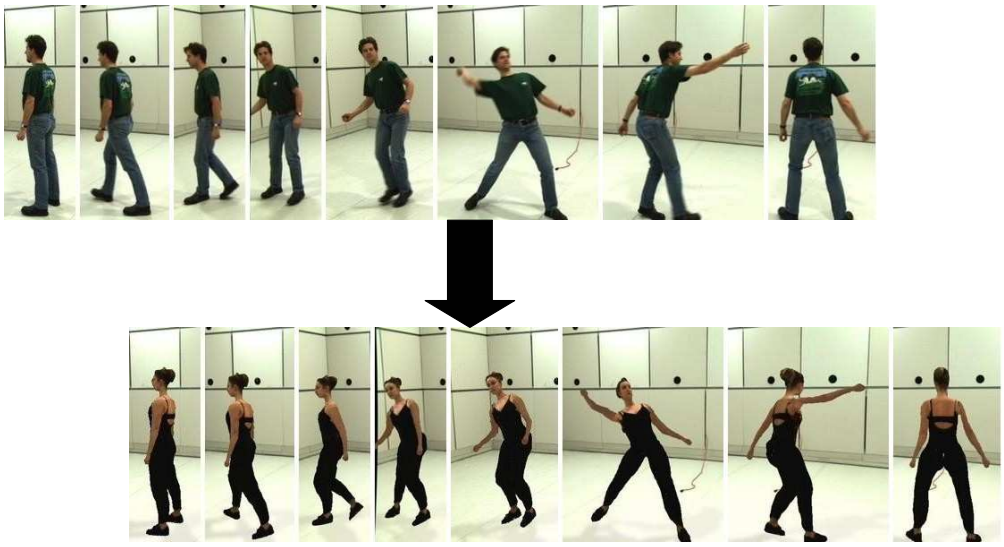
There are three major differences between the video-based motion transfer system described here from the other IBR algorithms. Firstly, the objects in the videos are not still objects but articulated human bodies. Moreover, the transfer is an “extrapolation” process which renders new motion to a different subject as opposed to the “interpolation” process which only replays motion of the same subject. Also the rendered pixels are taken from images across both time and space.



The rendering algorithm assumes there is a set of videos (with J frames and K cameras) of the subject (who is being rendered to perform the new motion) as source images. The motions in the source images are assumed to be tracked using the marker-less capture system. As a start, the new motion is applied to transform the articulated model. To render the motion from a target camera point of view, a ray is cast from a pixel from the image plane to intersect with the transformed model to locate the target model point P (Step 1). Using the tracked motion of the source data, the source model points P1 to PJ at each source frame are calculated (Step 2). These source points are then projected onto the corresponding source images and those projected pixels that are not occluded are taken as the source pixels (Step 3). Finally the source pixels are averaged with weights proportional to the viewing angles between the target camera and the source cameras.

Rendering Results

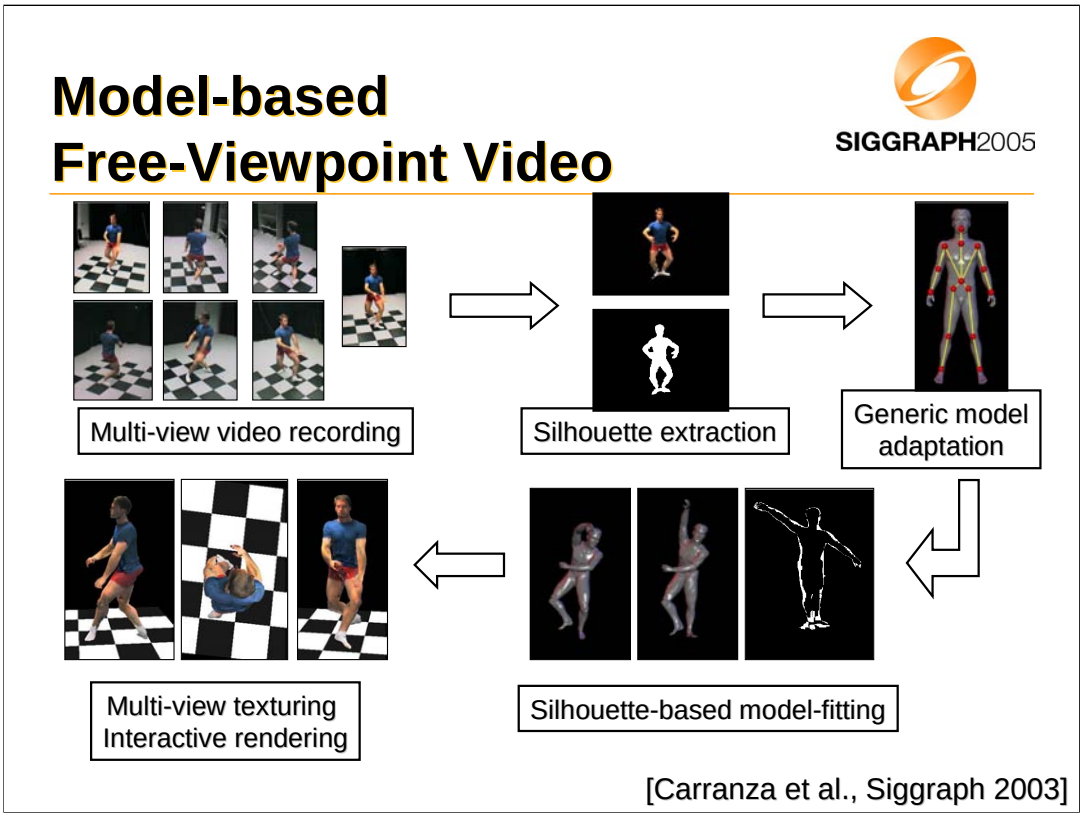
SIGGRAPH2005



The top image shows frames of a real sequence of a male subject miming a throw motion. The bottom image shows the corresponding frames of a photo-realistically synthesized sequence of a female subject performing the same motion. The throw motion is transferred from the male subject to the female subject using only vision-based algorithms. Details of the rendering algorithm and more results can be found in [Cheung et. al. 2004].

There are several limitations on this video-based motion transfer system. First of all, since the body parts are treated as rigid, smooth skinning is a major problem around joints. This also means that muscle deformation and clothing are not modeled. Artifacts are also caused by uneven scene lighting and self-shadows. Finally the appearance of the subject in the synthetic video depends solely on the images in the video source.

[Cheung et. al. 2004] K.M. Cheung, S. Baker, J. Hodgins and T. Kanade. Markerless Human Motion Transfer. *Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission*, September, 2004.



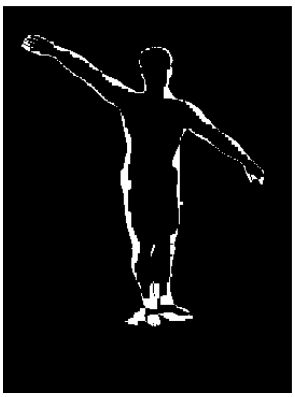
In free-viewpoint video the viewer is given the possibility to freely change his viewpoint onto a 3D rendition of a dynamic real-world scene. In order to generate a free-viewpoint video, the problems of acquisition of input data, reconstruction of a dynamic scene descriptions, and rendering in real-time have to be solved simultaneously.

This slide illustrates the workflow between algorithmic components of a model-based system to reconstruct and render free-viewpoint videos of human actors [Carranza et al., Siggraph'03]. Inputs to our system are multiple frame-synchronized video streams showing a moving person that have been captured with calibrated video cameras. Image silhouettes of the person in the foreground are extracted from each video frame. A generic human body model consisting of 16 segments, a triangle mesh surface geometry with roughly 21000 triangles, and a kinematic skeleton made of 17 joints is employed to represent the time-varying appearance of the human. An analysis-by-synthesis approach based on silhouette-overlap is used to adapt the model in shape and proportions to the actor, and to determine pose parameters for each time step of video. Real-time renditions of the captured scene are generated by projectively texturing the moving body model from the input video frames that are appropriately blended.

Silhouette Matching

SIGGRAPH2005

- Model Initialization & Marker-free Motion Capture
  - Non-linear minimization of pose/scaling parameters
  - Criterion: overlap between image silhouettes and body model projection
    - Area of intersection
    - Robust against silhouette inaccuracies
  - Graphics hardware acceleration
    - Pixel-wise XOR (stencil buffer)
    - 8 camera views / frame buffer read-write
    - 105 pose evaluations / sec. (GeForce 3)



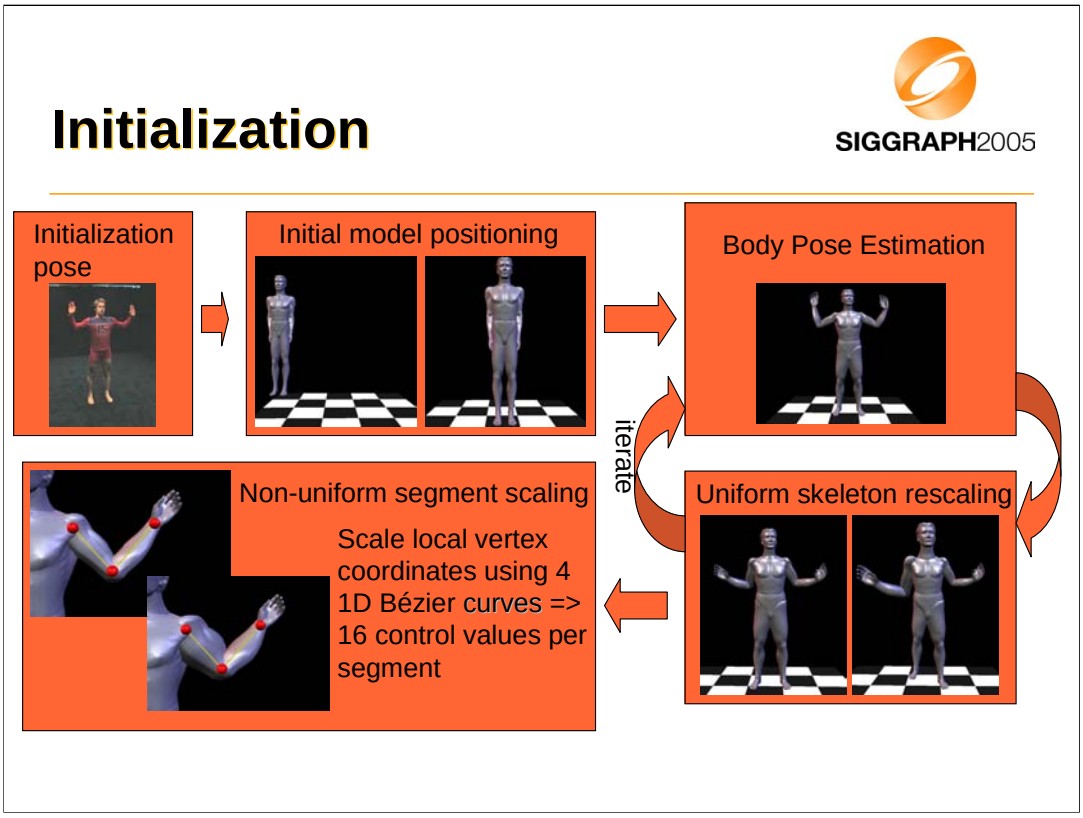
Silhouette XOR

Besides 35 parameters to specify the pose, our body model features for each segment a uniform scaling parameter, as well as 16 Bézier parameters for fine-tuning the appearance of the surface. Our analysis-by-synthesis approach employs the overlap between the projected model silhouettes and the image silhouettes in all camera views for two purposes:

1) Initialization: The geometry of the model is adapted to optimally represent the appearance of the real-world equivalent.

2) Marker-free Motion Capture: after the model has been customized, its pose is matched to the pose of the actor at each time step of the input video footage.

Both tasks require non-linear optimizations in the model parameters, the first one being performed in the pose and scaling parameters simultaneously, the second one being performed in the pose parameters only. The error function to be minimized computes the non-intersecting areas between the projected model and input silhouettes in all camera views. Conveniently, an estimate for this is obtained via the binary XOR between image and model silhouettes in all views. It can be efficiently evaluated on state-of-the-art graphics hardware using the stencil buffer. On a PC featuring an Intel Xeon 1.8 GHz CPU and an Nvidia GeForce3 GPU, we can perform 105 energy function evaluations using 8 cameras and a frame size of 320x240 pixels.



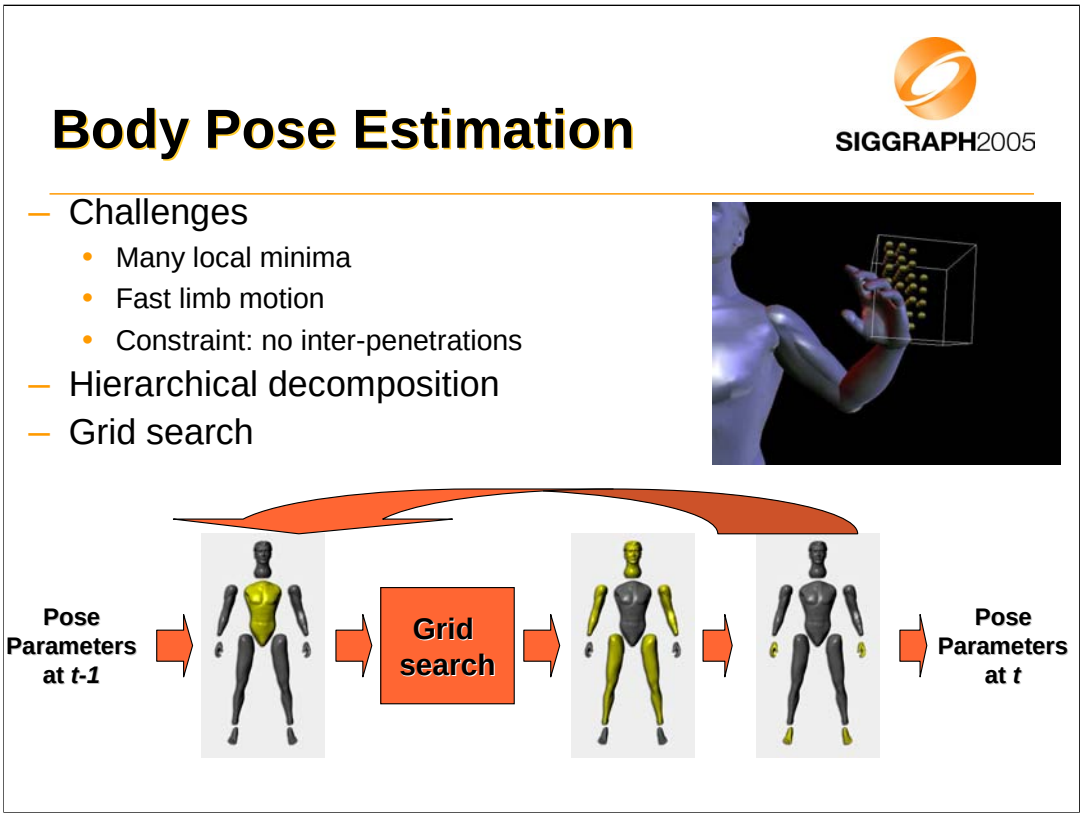
The shape and proportions of the body model are adapted to the shape of the actor in an iterative optimization procedure. Inputs are silhouette images of the actor striking a dedicated initialization pose. In a first step the position and orientation of the torso segment in 3D space is determined. The space of pose parameters for the torso is sampled regularly to find an optimal starting point for a subsequent downhill optimization that determines the final torso pose.

Thereafter, the method iterates between determining a new set of pose parameters for the whole model (explained on the next slide), and determining a new set of uniform scaling parameters for each segment.

Finally, optimal parameters for the 1D Bézier scaling functions are found that bring the segment outlines into optimal accordance with the multi-view silhouettes. On the hands and feet no Bézier scaling is performed.

The estimation of the optimal scaling parameters is performed hierarchically. It starts with the root of the skeleton (located in the torso) and proceeds layer-by-layer further down the skeleton hierarchy until the terminating nodes (head, hands and feet are reached).

For numerical minimization we employ a standard downhill direction set method such as Powell's method. The shape parameters of the model remain fixed for the duration of a free-viewpoint video.



Only with a completely passive marker-free motion capture approach the same video material can be used for motion and texture estimation. Determining the 35 pose parameters for each time step of video is a challenging problem since the non-convex energy-functional exhibits many local minima. Potentially rapid limb motion and constraints on allowable body poses require the parameter search space itself to be constrained. Only this way robust convergence to the correct solution can be assured.

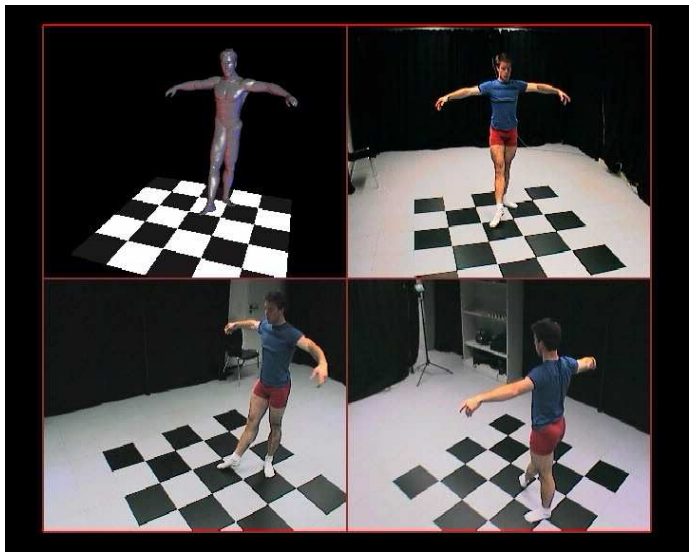
We initialize the optimization search for one time step t with the parameters found at t-1. The problem's search space is constrained by performing a hierarchical optimization that exploits structural knowledge about the body. The poses of individual kinematic sub-chains in the skeleton are determined separately. Body segments are considered in descending order with respect to the skeleton hierarchy and their respective influence on the silhouette appearance. In consequence, we first determine the torso pose, thereafter the poses of arms, legs, and head, and finally the parameters for the hands and feet.

For arms and legs we employ a custom 4-degree-of-freedom parameterization.

In order to handle rapid body motion the pose determination for each limb is preceded with a regular grid search in the 4D pose space. The best grid point found is used as starting point for the subsequent downhill optimization. Interpenetrating limb poses are also discarded during grid sampling.

Optionally, the complete pose estimation scheme is iterated.

Silhouette-based Motion Capture



The video shows three of the input camera views and the body model correctly following the motion of the dancer.

Model Fitting Acceleration

- Performance bottlenecks
  - Energy function
    - Transfer bandwidth (frame buffer read/write)
    - Rendering model geometry
  - Optimization
- Improvements
  - Modified XOR evaluation
  - Distributed Model Fitting

[Theobalt et al.; Vision, Modeling and Visualization, 2003]

The performance of the model fitting method is limited by two factors: the time needed to evaluate the XOR energy function and the runtime of the numerical minimizer itself.

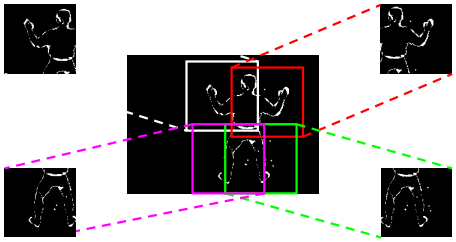
The performance of the energy function evaluation on the GPU is constrained by the overhead inflicted by the necessary frame-buffer read/write operations, as well as the overhead to render the model geometry.

The performance of the pose determination procedure is constrained by the fact that on a single PC one can only optimize the pose for one body segment at a time. Given more CPUs, however, the parameters for independent segments on the same level of the skeleton hierarchy could be efficiently estimated in parallel.

Thus, we have enhanced the GPU-based XOR computation in order to capitalize on the compartmentalized nature of the pose determination problem [Theobalt et al., VMV'03]. The implicit parallel structure of the problem also suggests a distributed implementation of the motion capture sub-system as a whole.

Accelerated Energy Function Evaluation

- XOR evaluation in sub-window



- Pre-rendering of non-moving body parts

Mask of unchanging body parts | **Output stencil with errors** | **Corrected energy function**

[Theobalt et al.; Vision, Modeling and Visualization, 2003]

We modify the error function evaluation in two ways in order to exploit the compartmentalized nature of the pose determination problem:

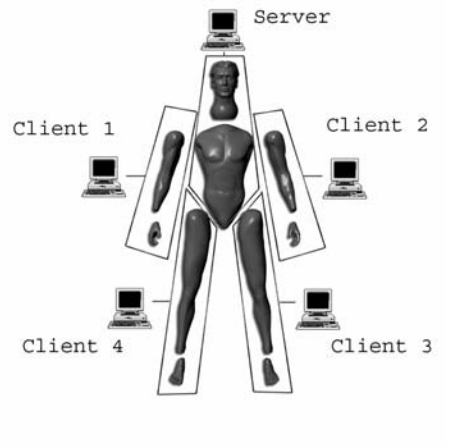
1) Instead of rendering the frames in full size, the rendering window for each camera views is confined to a limited area around the image plane location of that kinematic sub-chain which is currently optimized. By this means the amount of data that has to be transferred during frame-buffer read/write is significantly reduced.

2) The rendering overhead for the body model can be reduced if only the geometry of those body parts is displayed whose pose is currently optimized. The additional error in the XOR value in each camera view that is inflicted by not showing large parts of the model has to be compensated. We do this by applying an image-mask of unchanging body parts that is generated prior to the optimization. The bottom figure illustrates the process for one camera view.

Distributed Implementation

SIGGRAPH2005

- Client-server setup
- 5 PCs – 5 CPUs and 5 GPUs
- Fitting procedure
  - Server : torso
  - Clients/server in parallel: arms, legs, head
  - Clients in parallel: feet, hands



[Theobalt et al.; Vision, Modeling and Visualization, 2003]

Our distributed pose determination system is a client-server setup using 5 PCs, i.e. 5 CPUs and 5 GPUs. The hierarchical pose estimation procedure first determines the parameters of the torso on the server. The result is transferred to all clients. Now, the server and the clients determine the parameters of legs, arms and head in parallel. The results are broadcasted via the server before, in a final step, the poses of hands and feet are determined.

Results

- Average pose estimation time (s)

Method	Seq. A	Seq. B
XOR single computer	7.98	14.10
Sub-window, pre-render, single computer	3.30	10.10
Sub-window, pre-render, 5 computers	1.16	1.76

Xeon 1.8 Ghz , 512 MB RAM, GeForce3

The table shows average times needed for determining the pose of the body model at a single time step with the different algorithmic alternatives for silhouette fitting. The results we obtained with two test sequences are shown. Seq.A exhibits mostly slow body motion, whereas Seq.B shows an expressive jazz dance performance. Motion capture with the non-accelerated XOR computation on a single computer takes between 8s and 14s. A significant speed-up is obtained if we apply the enhanced energy function evaluation with sub-window constraint and body-part pre-rendering. By employing our distributed implementation we achieve average fitting times close to 1s.

Refined Model Fitting

- Silhouettes
  - Robustly detectable
  - Sensitive to large-scale motion
- Object texture
  - Highly detailed
  - Sensitive to small movements

⇒ Exploit texture for fine-tuning model parameters

The silhouette-based analysis-by-synthesis approach described on the previous slides enables us to reconstruct a dynamic scene description without having to modify the input video footage in any form, e.g. with optical markings in the scene. This is a necessary precondition if texture information is taken from the video streams as well.

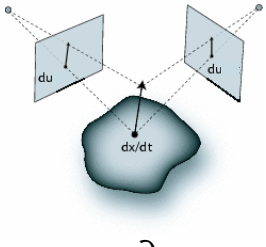
Image silhouettes can be computed robustly and our motion capture approach is fairly insensitive to measurement noise in the image data. However, although a silhouette-based approach robustly captures poses on a large scale, the exact pose of some body parts is hard to infer exactly. Slight pose inaccuracies can often be observed for those segments whose shape exhibits very few features on the silhouette outline that guide the optimization towards the correct parameters, such as the head.

We thus propose to enhance the original motion capture method into a hybrid approach that jointly uses silhouettes and texture information from the video frames for pose determination. The texture information enables the correction of slight pose inaccuracies that exist in the silhouette fit.

Texture-enhanced Motion Capture

SIGGRAPH2005

- 2D Optical Flow u_i
  - projection of 3D point motion $d\mathbf{x}/dt$ into 2D image plane i
- Optical Flow vs. Scene Flow
  - Jacobian matrix
    - Known from camera model
  - Optical Flow, Jacobian & Geometry
    - Solve for 3D motion $d\mathbf{x}/dt$



$$\mathbf{J}_i = \frac{\partial \mathbf{u}_i}{\partial x, y, z}$$

$$\mathbf{u}_i = \mathbf{J}_i \frac{d\mathbf{x}}{dt}$$

[Theobalt et al., Pacific Graphics, 2003]

Our texture-enhanced motion capture method employs the ***scene flow***, the 3D equivalent of the 2D optical flow in the image plane, to compute corrective pose updates.

The 2D optical flow is the projection of the 3D motion field of a dynamic scene into the image plane of a recording camera.

A sea of algorithms has been proposed in the computer vision literature to compute this 2D flow field between two subsequent depictions of the scene. In our implementation we employ the method by Lukas and Kanade (see [Theobalt et al., PG'03] for references to the original literature).

The scene flow corresponding to a set of optical flows in multiple camera views is the set of 3D motion vectors, one for each point in the scene, whose projections are the 2D optical flows. The differential relationship between the optical flow and the scene flow is described by a Jacobian matrix whose entries can be determined from the matrix of a calibrated camera.

Given a set of optical flows from multiple calibrated camera views, and full knowledge about the 3D geometry, it is possible to infer the scene flow vector for each point on the geometry by solving a linear equation system.

Motion Field-guided Model Fitting Algorithm

- Estimate model pose for t+1
  - Silhouette fitting
- Render image estimates $I'_{j,t+1}$
  - Texture model with images $I_{j,t}$
- For each model vertex
  - Determine projection coordinates and visibility in each image
  - Compute optical flow between $I_{j,t+1}$ and $I'_{j,t+1}$
  - Calculate 3D vertex motion from motion field
- Update model to conform with motion field

In a predictor-corrector-scheme, we employ the scene flow determination method for computing differential pose updates that correct inaccuracies in the silhouette-fit.

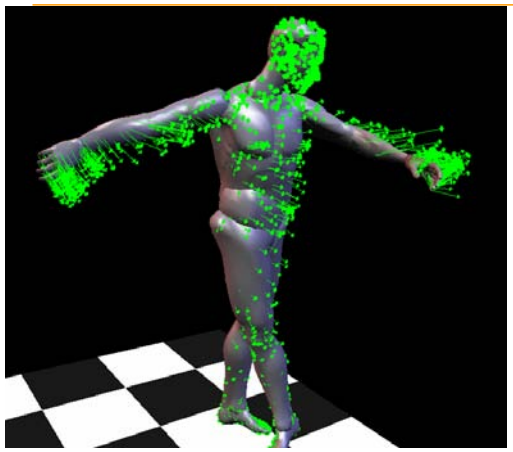
In a first step, an estimate of the pose parameters at time t+1 is computed using the original silhouette-based motion capture technique.

Using this first pose estimate, we predict the appearance of the model by rendering and projectively texturing it with the camera images from the previous time step $I_{j,t}$. This way, we generate novel images $I'_{j,t+1}$ showing the predicted model appearance at time t+1 in each camera view j.

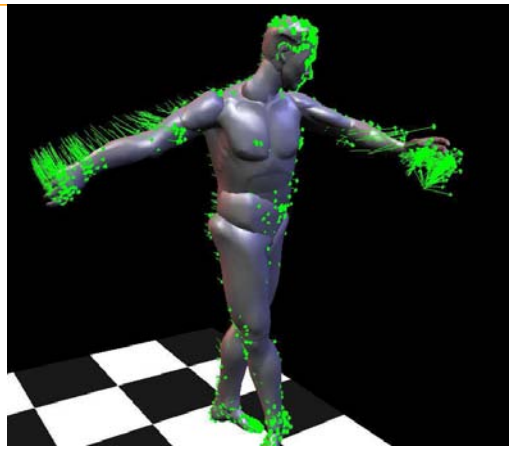
For each vertex it is determined in which cameras it is visible. For each camera that sees the vertex, the optical flow between its projection into $I'_{j,t+1}$ and $I_{j,t+1}$, is computed. Using all the 2D flow vectors found for this vertex, the corresponding 3D scene flow vector is computed using the method outlined on the previous slide.

This way, a scene flow field is generated which describes for each vertex a position update that brings the model into a pose that is photo-consistent with all camera images. The corrective flow field is translated into a set of corrective pose parameters for the model by means of a shape registration method originally proposed by Horn (see [Theobalt et al., PG'03]).

Differential Pose Update



Before scene-flow-based pose update



After registering torso, head, and upper arm and upper leg

3D flow vectors purposefully exaggerated

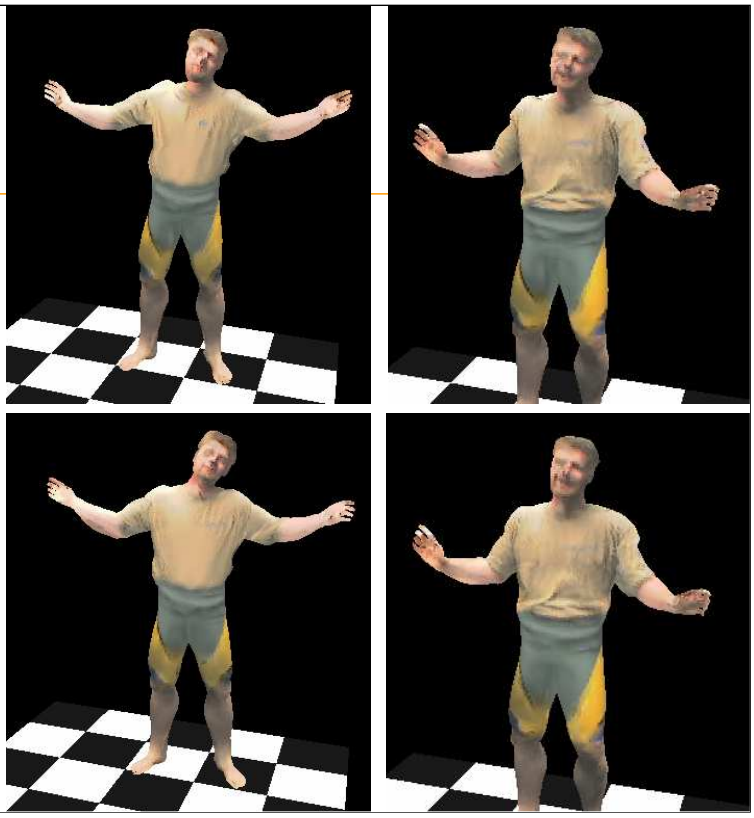
The figure on the left shows the body model in the pose that was found via silhouette–fitting.

The green arrows are corrective scene flow vectors that have been computed for all vertices in the body model using the predictor-corrector scheme described on the previous slide. The length of the flow vectors has been purposefully exaggerated in order to better visualize the flow field.

The figure on the right shows the pose of the model after the corrective pose update parameters for the torso, the head, the upper arms and the upper legs have been applied to the skeleton.

Results

- Purely silhouette-based pose determination
- Pose determination with differential update



The four figures illustrate the visual improvements achievable with the texture-enhanced motion capture method. The top row shows screen-shots of free-viewpoint videos that have been reconstructed with pure silhouette-fitting.

The bottom row shows renditions from the same virtual camera views but with the scene-flow-based pose correction applied. Improvements are mainly visible in the face and on the torso. Block artifacts in the images are due to the limited camera resolution of 320x240 pixels.

Rendering

- Render model geometry in captured pose
- Dynamic projective texturing with camera images
  - Per-vertex blending and interpolation in fragments
    - Texture information (camera image) $tex_j(i)$
    - Visibility $Vis_j(i)$
    - View-independent spatial blending weights $\omega_j(i)$
    - Optional: view-dependent weights $\rho_j(i)$

Final color $c_i = \sum_j^N Vis_j(i) \cdot \rho_j(i) \cdot \omega_j(i) \cdot tex_j(i)$

The reconstructed 3D videos are rendered by displaying the body model in the sequence of captured poses, and projectively texturing it with the video frames at each time step of video.

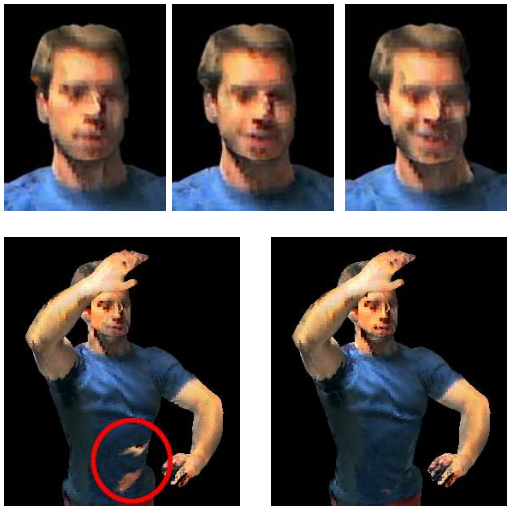
For vertex *i* in the model we compute the final color c_i by appropriately weighting and summing the color contributions from each input camera view $tex_j(i)$.$Vis_j(i)$ is the visibility of vertex *i* in camera view *j*, $\omega_j(i)$ is a spatial blending weight depending on the relative orientation of the vertex with respect to the input camera. We can assume that the reflectance of most garments is close to lambertian. It is thus valid to compute the spatial texture blending weights only based on the orientation of a vertex with respect to the input cameras. The camera which sees a surface element most head-on is assigned the highest blending weight.

However, in order to model view-dependent reflectance effects appearing in some garments, an optional view-dependent rescaling factor $\rho_j(i)$ can be included into the color computation. It weights the camera contributions depending on the relation between outgoing viewing direction and camera viewing direction. The per-vertex blending weights are interpolated in the fragment stage of the GPU.

Video Texturing

SIGGRAPH2005

- Time-varying texture
  - Cloth folds
  - Local shadows
  - Facial expressions
- Detail preservation
  - Small inaccuracies between geometry and silhouettes
  - Soft shadow visibility
  - Camera image dilation
  - Controllable spatial blending weights



By generating a novel texture for each time step of video, subtle dynamic details in surface appearance, such as wrinkles in clothing, local shadows and facial expressions are faithfully reproduced (see three Figs. In top row, block artifacts are due to the limited camera resolution of 320x240).

Although our automatic model initialization approach generates a body geometry that matches the appearance of the actor very well, small geometry misalignments between the virtual and the real human may still exist. In These mismatches may lead to erroneous projections of parts of the texture belonging to occluding geometry onto actually more distant parts of the body (bottom left Fig.). Furthermore, seams originating from projected silhouette boundaries may appear on the body.

We employ three techniques to counter these effects. Firstly, we compute the visibility of each vertex from a set of slightly displaced camera views (soft shadow visibility). This way, projection artifacts at occlusion boundaries are prevented (bottom right Fig.). Secondly, we dilate the segmented video frames at the silhouette boundaries to prevent the seams. Thirdly, we apply a special method to compute controllable spatial texture blending weights which is illustrated on the next slide.

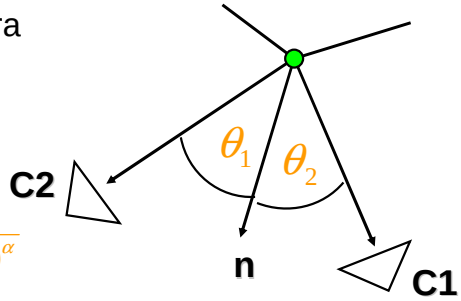
Controllable View-independent Spatial Texture Blending

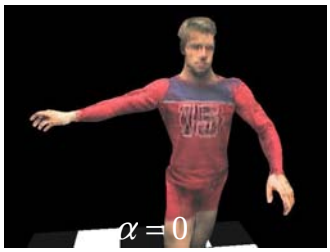
SIGGRAPH2005

– dependent on surface-to-camera orientation

$$\omega_j(i) = \frac{1}{\Theta_j(i)}$$

$$\omega'_j(i) = \frac{1}{(\max_k(\omega_k(i)) + 1 - \omega_j(i))^{\alpha}}$$





The easiest way to compute a view-independent spatial blending weight for vertex *i* and camera *j*, $\omega_j(i)$, is to apply the reciprocal of the angle $\theta_j(i)$ between the vertex normal and the viewing vector towards the camera.

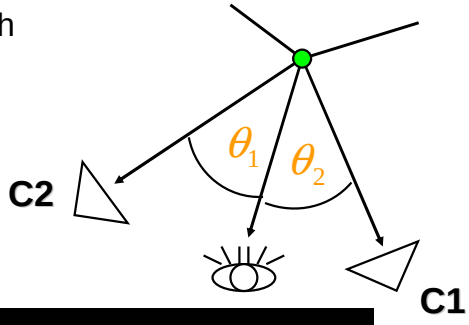
We propose alternative spatial blending weights, $\omega'_j(i)$, that give a better control over the influence of each camera on the appearance of the final texture. The alternative weight computation assigns a proportionally high weight to the camera which sees the vertex most head-on. The sharpness value α controls the amount by which the influence of the best camera is exaggerated. In the limit, $\alpha \to \infty$, only the best camera is contributing to the color of the vertex. The three figures on the bottom of the slide illustrate the influence of the sharpness value on the final renditions.

View-dependent Rescaling Weights

SIGGRAPH2005

- Dependent on orientation with respect to output viewpoint

$$\rho_j(i) = \frac{1}{\Theta_j(i)}$$



View-independent blending only

View-dependent rescaling applied

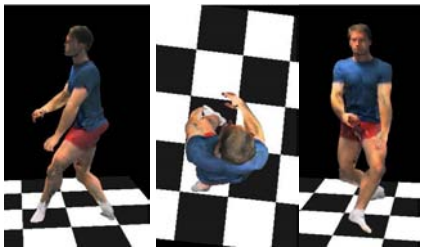
Optionally, a weight $\rho_j(i)$ can be computed that view-dependently rescales the view-independent blending weights.

The weight $\rho_j(i)$ for vertex i and camera j is the reciprocal of the angle $\Theta_j(i)$ between the viewing direction towards the camera and the current outgoing viewing direction.

Free-Viewpoint Video Viewer

SIGGRAPH2005

– Rendering: 30 fps (GeForce3)

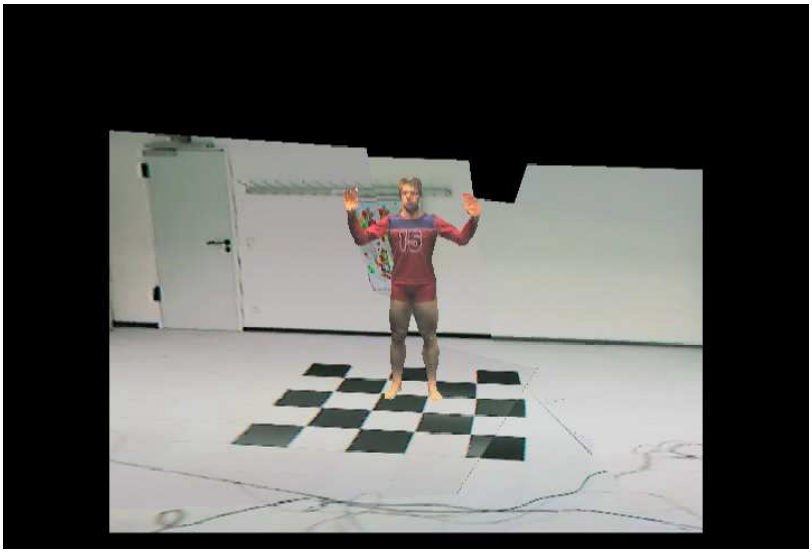


The top right image shows a screen-shot of our renderer which displays free-viewpoint videos at 30 fps on an Nvidia GeForce3 GPU.

The bottom row shows screen-shots of free-viewpoint videos that have been rendered from virtual camera views different from any input camera view.

The input video footage for these sequences (as well as all the other sequences shown on previous slides) has been captured with 8 video cameras that provide an image resolution of 320x240 pixels each.

Result



A free-viewpoint video of ballet dancer that we have reconstructed with our method.

References
(not complete !)

Small-baseline VBR

- 3D TV
  - [Matusik & Pfister, Siggraph'04]
- Dynamic light field warping
  - [Goldluecke & Magnor, VMV'02]
  - [Zitnick et al., Siggraph'04]

Depth-from-Stereo

- Pairwise Stereo
  - Fast [Atzpatin et al., T-CSVT'04], [Criminisi & Blake, CVPR'04]
- Multi-view
  - Graph cuts [Kolmogorov & Zabih, ECCV'02]
  - With segmentation [Goldluecke & Magnor, CVPR'03]
  - On GPU [Yang et al. PG'02], [Yang and Pollefeys CVPR'03], [Yang et al. ICCV'03], [Yang and Pollefeys RTI'05]
- In space-time
  - Active triangulation [Zhang et al., CVPR'03], [Davis et al., CVPR'03]
- With opacity
  - Transparency & matting [Szeliski & Golland, ICCV'98]
  - Opacity hulls [Matusik et al., Siggraph'02]
- Overview
  - [Scharstein & Szeliski, IJCV'02],[Szeliski & Zabih, LNCS 1883, '99]

Wide-baseline VBR

- Visual hull
  - Volumetric [Potmesil, CVGIP'87],[Szeliski, CVGIP'93]
  - Image-based [Matusik et al., Siggraph'00]
  - Polygonal [Matusik et al., EG'01],[Franco&Boyer, BMVC'03]
  - On hardware [Li et al., GI'04]
- Photo hull
  - Voxel coloring [Seitz & Dyer, CVPR'97]
  - Space carving [Kutulakos & Seitz, ICCV'99]
  - Roxels [De Bonet & Viola, ICCV'99]
  - On GPU [Slabaugh et al., 3DPVT'02,[Li et al., EG'04]
- Silhouettes and photo-consistency
  - [Sinha and Pollefeys, submitted to ICCV'05]

Camera network calibration

- Silhouette-based camera network calibration

  [Sinha et al,CVPR'04][Sinha and Pollefeys,ICPR'04]
  [Sinha and Pollefeys, 3DPVT'04]

Spacetime Coherence

- Scene flow
  - Scene flow [Vedula et al., ICCV'99, CVPR'00]
  - Surfel sampling [Carceroni & Kutulakos, ICCV'01]
  - Spacetime trajectories [Neumann & Aloimonos, IJCV'02]
  - Visual hull alignment & refinement [Cheung et al., CVPR'03]
- Spacetime-coherent reconstruction
  - Theory [Goldluecke & Magnor, ECCV'04]
  - Practice [Goldluecke & Magnor, CVPR'04]

Model-based VBR

- A-priori model usage
  - Fitting 3D parameterized model [Lowe, PAMI'91]
  - Force field, distance maps [Brunie et al. ECCV'92]
  - Photo registration [Neugebauer & Klein, CGF'99]
- Analysis-by-synthesis
  - Model pose [Koch, PAMI'93], [Eisert et al., T-CSVT'00]
  - Camera calibration [Eisert et al., VMV'02]
  - Photo registration [Lensch et al., PG'00]
  - Motion capture [Carranza et al., Siggraph'03]
- Free-Viewpoint Video
  - Silhouette-based motion capture [Carranza et al., Siggraph'03]
  - Texture-based refinement [Theobalt et al., PG'03]
  - Analysis in parallel [Theobalt et al., VMV'03]

VBR Resources

www.video-based-rendering.org

Presenters

- German Cheung
  - gcheung@nevenvision.com
- Marcus Magnor
  - magnor@mpi-sb.mpg.de
- Wojciech Matusik
  - matusik@merl.com
- Marc Pollefeys
  - marc@cs.unc.edu
- Christian Theobalt
  - theobalt@mpi-sb.mpg.de