# Point-Based Computer Graphics

Eurographics 2003 Tutorial T1

**Organizers**

Markus Gross
ETH Zürich

Hanspeter Pfister
MERL, Cambridge

**Presenters**

Marc Alexa
TU Darmstadt

Carsten Dachsbacher
Universität Erlangen-Nürnberg

Markus Gross
ETH Zürich

Mark Pauly
ETH Zürich

Jeroen van Baar
MERL, Cambridge

Matthias Zwicker
ETH Zürich

## Contents



## Tutorial Schedule

Introduction (Markus Gross)
Acquisition of Point-Sampled Geometry and Appearance (Jeroen van Baar)
Point-Based Surface Representations (Marc Alexa)
Point-Based Rendering (Matthias Zwicker)

*Lunch*

Sequential Point Trees (Carsten Dachsbacher)
Efficient Simplification of Point-Sampled Geometry (Mark Pauly)
Spectral Processing of Point-Sampled Geometry (Markus Gross)
Pointshop3D: A Framework for Interactive Editing of Point-Sampled Surfaces (Markus Gross)
Shape Modeling (Mark Pauly)
Pointshop3D Demo (Mark Pauly)
Discussion (all)

## Presenters and Organizers Contact Information

**Dr. Markus Gross**
Professor
Department of Computer Science
Swiss Federal Institute of Technology (ETH)
CH 8092 Zürich
Switzerland
Phone: +41 1 632 7114
FAX: +41 1 632 1596
grossm@inf.ethz.ch
http://graphics.ethz.ch

**Dr. Hanspeter Pfister**
Associate Director
MERL - A Mitsubishi Electric Research Lab
201 Broadway
Cambridge, MA 02139
USA
Phone: (617) 621-7566
Fax: (617) 621-7550
pfister@merl.com
http://www.merl.com/people/pfister/

**Jeroen van Baar**
MERL - A Mitsubishi Electric Research Lab
201 Broadway
Cambridge, MA 02139
USA
Phone: (617) 621-7577
Fax: (617) 621-7550
jeroen@merl.com
http://www.merl.com/people/jeroen/

**Matthias Zwicker**
Department of Computer Science
Swiss Federal Institute of Technology (ETH)
CH 8092 Zürich
Switzerland
Phone: +41 1 632 7437
FAX: +41 1 632 1596
zwicker@inf.ethz.ch
http://graphics.ethz.ch

**Mark Pauly**
Department of Computer Science
Swiss Federal Institute of Technology (ETH)
CH 8092 Zürich

Switzerland
Phone: +41 1 632 0906
FAX: +41 1 632 1596
pauly@inf.ethz.ch
http://graphics.ethz.ch

**Dr. Marc Stamminger**
Universität Erlangen-Nürnberg
Am Weichselgarten 9
91058 Erlangen
Germany
Phone: +49 9131 852 9919
FAX: +49 9131 852 9931
Marc.Stamminger@informatik.uni-erlangen.de

**Carsten Dachsbacher**
Universität Erlangen-Nürnberg
Am Weichselgarten 9
91058 Erlangen
Germany
Phone: +49 9131 852 9925
FAX: +49 9131 852 9931
dachsbacher@informatik.uni-erlangen.de

**Dr. Marc Alexa**
Interactive Graphics Systems Group
Technische Universität Darmstadt
Fraunhoferstr. 5
64283 Darmstadt
Germany
Phone: +49 6151 155 674
FAX: +49 6151 155 669
alexa@gris.informatik.tu-darmstadt.de
http://www.dgm.informatik.tu-darmstadt.de/staff/alexa/

## References

M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. Silva. Point set surfaces. Proceedings of IEEE Visualization 2001, p. 21-28, San Diego, CA, October 2001.

C. Dachsbacher, C. Vogelsang, M. Stamminger, Sequential point trees. Proceedings of SIGGRAPH 2003, *to appear*, San Diego, CA, July 2003.

O. Deussen, C. Colditz, M. Stamminger, G. Drettakis, Interactive visualization of complex plant ecosystems. Proceedings of IEEE Visualization 2002, Boston, MA, October 2002.

W. Matusik, H. Pfister, P. Beardsley, A. Ngan, R. Ziegler, L. McMillan, Image-based 3D photography using opacity hulls. Proceedings of SIGGRAPH 2002, San Antonio, TX, July 2002.

W. Matusik, H. Pfister, A. Ngan, R. Ziegler, L. McMillan, Acquisition and rendering of transparent and refractive objects. Thirteenth Eurographics Workshop on Rendering, Pisa, Italy, June 2002.

M. Pauly, R. Keiser, L. Kobbelt, M. Gross, Shape modelling with point-sampled geometry, *to appear*, Proceedings of SIGGRAPH 2003, San Diego, CA, July 2003.

M. Pauly, M. Gross, Spectral processing of point-sampled geometry. Proceedings of SIGGRAPH 2001, p. 379-386, Los Angeles, CA, August 2001.

M. Pauly, M. Gross, Efficient Simplification of Point-Sampled Surfaces. IEEE Proceedings of Visualization 2002, Boston, MA, October 2002.

H. Pfister, M. Zwicker, J. van Baar, M. Gross, Surfels - surface elements as rendering primitives. Proceedings of SIGGRAPH 2000, p. 335-342, New Orleans, LS, July 2000.

M. Stamminger, G. Drettakis, Interactive sampling and rendering for complex and procedural geometry, Rendering Techniques 2001, Proceedings of the Eurographics Workshop on Rendering 2001, June 2001.

L. Ren, H. Pfister, M. Zwicker, Object space EWA splatting: a hardware accelerated approach to high quality point rendering. Proceedings of the Eurographics 2002, to appear, Saarbrücken, Germany, September 2002.

M. Zwicker, H. Pfister, J. van Baar, M. Gross, EWA volume splatting. Proceedings of IEEE Visualization 2001, p. 29-36, San Diego, CA, October 2001.

M. Zwicker, H. Pfister, J. van Baar, M. Gross, Surface splatting. Proceedings of SIGGRAPH 2001, p. 371-378, Los Angeles, CA, August 2001.

M. Zwicker, H. Pfister, J. van Baar, M. Gross, EWA splatting. IEEE Transactions on Visualization and Computer Graphics.

M. Zwicker, M. Pauly, O. Knoll, M. Gross, Pointshop 3D: an interactive system for point-based surface editing. Proceedings of SIGGRAPH 2002, San Antonio, TX, July 2002

**Project Pages**

- Rendering
  http://graphics.ethz.ch/surfels
- Acquisition
  http://www.merl.com/projects/3Dimages/

- Sequential point trees
  http://www9.informatik.uni-erlangen.de/Persons/Stamminger/Research
- Modeling, processing, sampling and filtering
  http://graphics.ethz.ch/points
- Pointshop3D
  http://www.pointshop3d.com

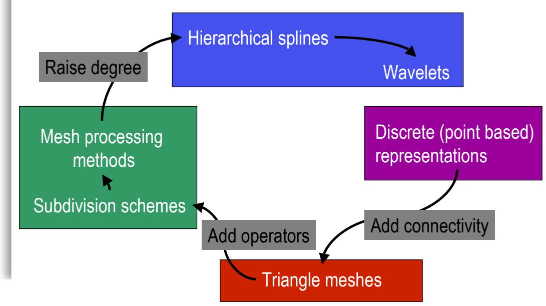
## Eurographics 2003

# Point-Based Computer Graphics

Tutorial T1

Marc Alexa, Carsten Dachsbacher,
Markus Gross, Mark Pauly,
Hanspeter Pfister, Marc Stamminger,
Jeroen Van Baar, Matthias Zwicker

## Surf. Reps. for Graphics






## Polynomials...

- ✓ Rigorous mathematical concept
- ✓ Robust evaluation of geometric entities
- ✓ Shape control for smooth shapes
- ✓ Advanced physically-based modeling

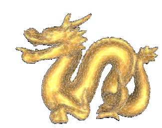
- × Require parameterization
- × Discontinuity modeling
- × Topological flexibility

***Refine h rather than p !***



## Polynomials -> Triangles

- Piecewise linear approximations
- Irregular sampling of the surface
- Forget about parameterization



Triangle meshes

- Multiresolution modeling
- Compression
- Geometric signal processing



## Triangles...

- ✓ Simple and efficient representation
- ✓ Hardware pipelines support Δ
- ✓ Advanced geometric processing is being in sight
- ✓ The widely accepted queen of graphics primitives

- × Sophisticated modeling is difficult
- × (Local) parameterizations still needed
- × Complex LOD management
- × Compression and streaming is highly non-trivial

***Remove connectivity !***



## Triangles -> Points

- From piecewise linear functions to Delta distributions
- Forget about connectivity

Point clouds

- Points are natural representations within 3D acquisition systems
- Meshes provide an articifical enhancement of the acquired point samples

## History of Points in Graphics

- Particle systems [Reeves 1983]
- Points as a display primitive [Whitted, Levoy 1985]
- Oriented particles [Szeliski, Tonnesen 1992]
- Particles and implicit surfaces [Witkin, Heckbert 1994]
- Digital Michelangelo [Levoy et al. 2000]
- Image based visual hulls [Matusik 2000]
- Surfels [Pfister et al. 2000]
- QSplat [Rusinkiewicz, Levoy 2000]
- Point set surfaces [Alexa et al. 2001]
- Radial basis functions [Carr et al. 2001]
- Surface splatting [Zwicker et al. 2001]
- Randomized z-buffer [Wand et al. 2001]
- Sampling [Stamminger, Drettakis 2001]
- Opacity hulls [Matusik et al. 2002]
- Pointshop3D [Zwicker, Pauly, Knoll, Gross 2002]...?



## The Purpose of our Course is …

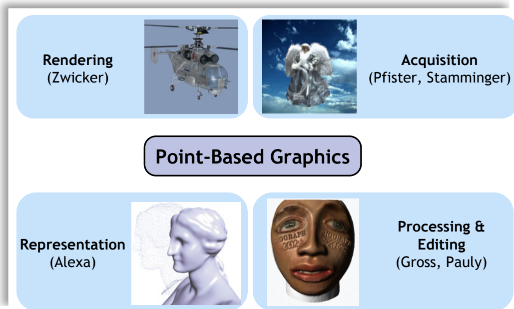
*I) …to introduce points as a versatile and powerful graphics primitive*

*II) …to present state of the art concepts for acquisition, representation, processing and rendering of point sampled geometry*

*III) …to stimulate YOU to help us to further develop Point Based Graphics*



## Taxonomy






## Morning Schedule

- Introduction (Markus Gross)
- Acquisition of Point-Sampled Geometry and Apprearance (Jeroen van Baar)
- Point-Based Surface Representations (Marc Alexa)
- Point-Based Rendering (Matthias Zwicker)



## Afternoon Schedule

- Sequential point trees (Carsten Dachsbacher)
- Efficient simplification of point-sampled geometry (Mark Pauly)
- Spectral processing of point-sampled geometry (Markus Gross)
- Pointshop3D: A framework for interactive editing of point-sampled surfaces (Markus Gross)
- Shape modeling (Mark Pauly)
- Pointshop3D demo (Mark Pauly)
- Discussion (all)

# Eurographics 2003

## Acquisition of Point-Sampled Geometry and Appearance


Jeroen van Baar and Hanspeter Pfister, MERL
[jeroen,pfister]@merl.com

Wojciech Matusik, MIT
Addy Ngan, MIT
Paul Beardsley, MERL
Remo Ziegler, MERL
Leonard McMillan, MIT




## The Goal: To Capture Reality

- Fully-automated 3D model creation of real objects.
- Faithful representation of appearance for these objects.



## Image-Based 3D Photography

- An image-based 3D scanning system.
  - Handles fuzzy, refractive, transparent objects.
  - Robust, automatic
  - Point-sampled geometry based on the *visual hull*.
  - Objects can be rendered in novel environments.



## Previous Work

- Active and passive 3D scanners
  - Work best for diffuse materials.
  - Fuzzy, transparent, and refractive objects are difficult.
- BRDF estimation, inverse rendering
- Image based modeling and rendering
  - Reflectance fields [Debevec et al. 00]
    - Light Stage system to capture reflectance fields
    - Fixed viewpoint, no geometry
  - Environment matting [Zongker et al. 99, Chuang et al. 00]
    - Capture reflections and refractions
    - Fixed viewpoint, no geometry

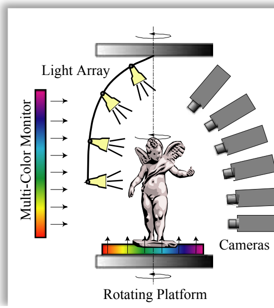


## Outline

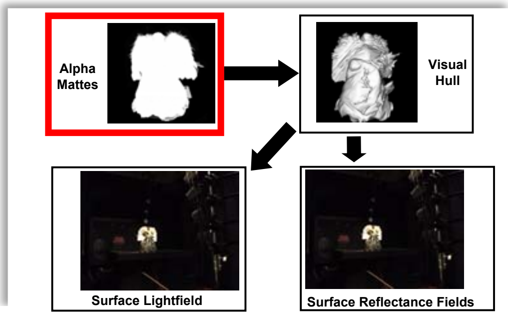
- Overview
- ➢ System
- Geometry
- Reflectance
- Refraction & Transparency



## Acquisition System

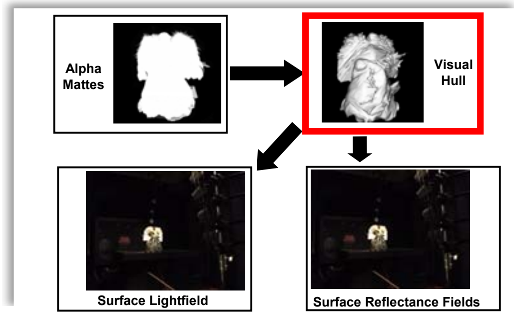



## Acquisition Process



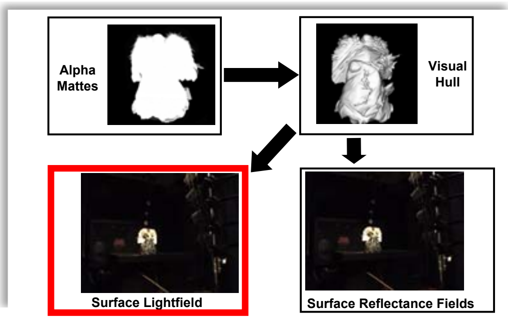



## Acquisition Process



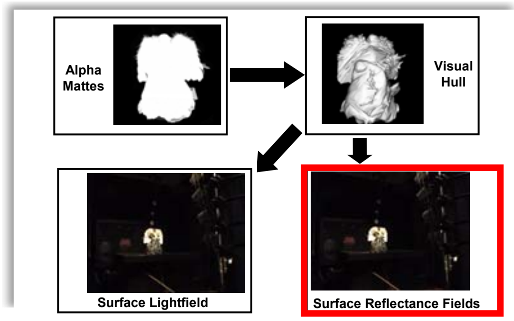



## Acquisition Process






## Acquisition Process






## Outline

- Overview
- System
- ➢ Geometry
- Reflectance
- Refraction & Transparency

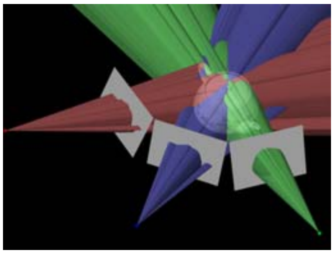


## Acquisition

- For each viewpoint ( 6 cameras x 72 positions )
  - Alpha mattes
    - Use multiple backgrounds [Smith and Blinn 96]
  - Reflectance images
    - Pictures of the object under different lighting (4 lights x 11 positions)
  - Environment mattes
    - Use similar techniques as [Chuang et al. 2000]

## Geometry – Opacity Hull

- Visual hull: The maximal object consistent with a given set of silhouettes.





## Geometry Example



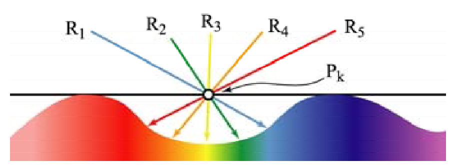
## Approximate Geometry

- The approximate visual hull is augmented by radiance data to render concavities, reflections, and transparency.



## Surface Light Fields

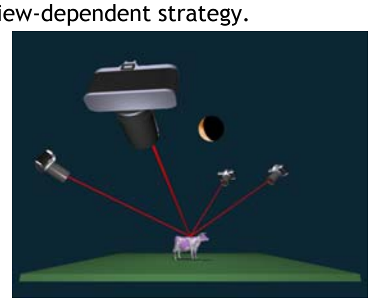
- A surface light field is a function that assigns a color to each ray originating on a surface. [Wood *et al.*, 2000]






## Shading Algorithm

- A view-dependent strategy.





## Color Blending

- Blend colors based on angle between virtual camera and stored colors.
- Unstructured Lumigraph Rendering
  [Buehler et al., SIGGRAPH 2001]
- View-Dependent Texture Mapping
  [Debevec, EGRW 98]

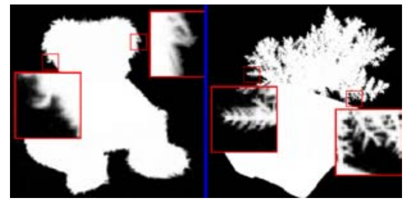
## Point-Based Rendering

- Point-based rendering using LDC tree, visibility splatting, and view-dependent shading.



## Geometry – Opacity Hull

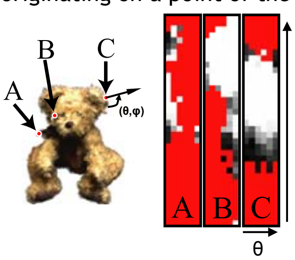
- Store the opacity of each observation at each point on the visual hull [Matusik et al. SIG2002].





## Geometry – Opacity Hull

- Assign view-dependent opacity to each ray originating on a point of the visual hull.




Red = invisible
White = opaque
Black = transparent



## Example

Photo



## Example

Photo

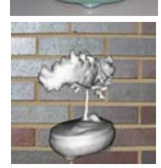
Visual Hull



## Example

Photo

Visual Hull



Opacity Hull

## Example

Photo

Surface Light Field

Visual Hull

Opacity Hull



## Results

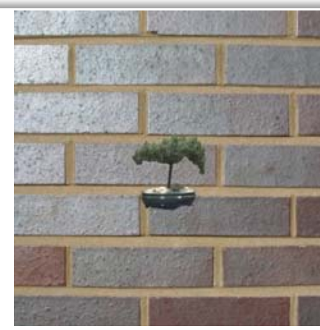
- Point-based rendering using EWA splatting, A-buffer blending, and edge antialiasing.



## Results Video



## Results Video

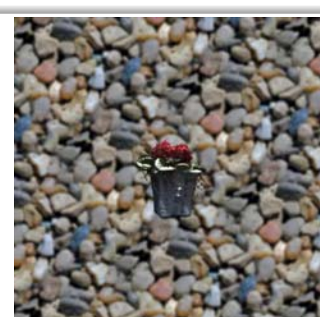




## Results Video



## Results Video



## Opacity Hull - Discussion

- View dependent opacity vs. geometry trade-off.
- Sometimes acquiring the geometry is not possible.
- Sometimes representing true geometry would be very inefficient.
- Opacity hull stores the "macro" effect.



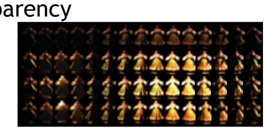
## Point-Based Models

- No need to establish topology or connectivity.
- No need for a consistent surface parameterization for texture mapping.
- Represent organic models (feather, tree) much more readily than polygon models.
- Easy to represent view-dependent opacity and radiance per surface point.



## Outline

- Overview
- Previous Works
- Geometry
- ➢ Reflectance
- Refraction & Transparency





## Light Transport Model

- Assume illumination originates from infinity.
- The light arriving at a camera pixel can be described as:

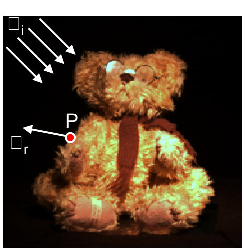
$$C(x,y) = \int_{\Omega} W(\omega) E(\omega) d\omega$$

*C(x,y)* - the pixel value
*E* - the environment
*W* - the *reflectance field*



## Surface Reflectance Fields

- 6D function: $W(P,\omega_i,\omega_r) = W(u_r,v_r;\theta_i,\Phi_i;\theta_r,\Phi_r)$

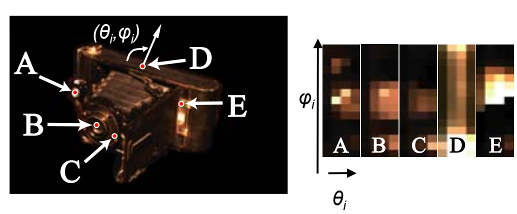




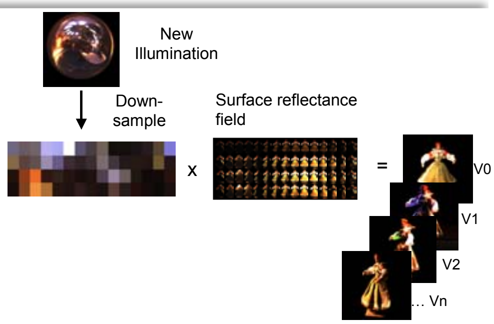

## Reflectance Functions

- For each viewpoint, 4D function:

$$W_{xy}(\omega_i) = W(x,y;\theta_i,\Phi_i)$$



## Relighting



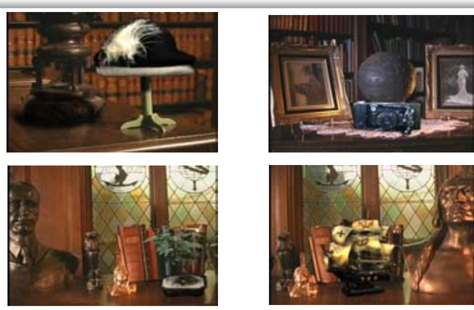



## Compression

- Subdivide images into 8 x 8 pixel blocks.
- Keep blocks containing the object (avg. compression 1:7)
- PCA compression (avg. compression 1:10)

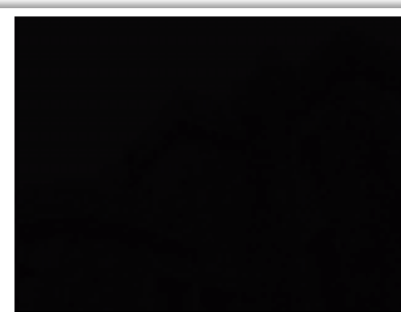



## Results





## The Library





## Surface Reflectance Fields

- Work without accurate geometry
- Surface normals are not necessary
- Capture more than reflectance
  - Inter-reflections
  - Subsurface scattering
  - Refraction
  - Dispersion
  - Non-uniform material variations
- Simplified version of the BSSRDF

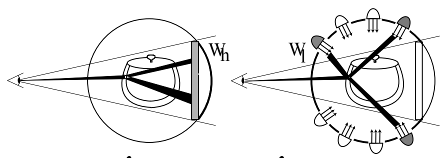


## Outline

- Overview
- Previous Works
- Geometry
- Reflectance
- ➢ Refraction & Transparency

## Acquisition

- We separate the hemisphere into high resolution □$_h$ and low resolution □$_l$.




$$C(x,y) = \int_{\Omega_h} W_h(\xi)T(\xi)d\xi + \int_{\Omega_l} W_l(\omega_i)L(\omega_i)d\omega$$



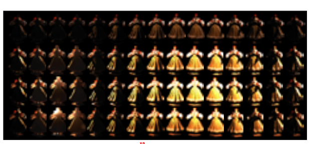
## Acquisition

- For each viewpoint ( 6 cameras x 72 positions )
  - Alpha mattes
    - Use multiple backgrounds [Smith and Blinn 96]
  - Reflectance images ← Low resolution
    - Pictures of the object under different lighting (4 lights x 11 positions)
  - Environment mattes ← High resolution
    - Use similar techniques as [Chuang et al. 2000]



## Low-Resolution Reflectance Field

$$C(x,y) = \int_{\Omega_h} W_h(\xi)T(\xi)d\xi + \int_{\Omega_l} W_l(\omega_i)L(\omega_i)d\omega$$



$$\int_{\Omega_l} W_l(\omega_i)L(\omega_i)d\omega \approx \sum_{i=1}^{n} W_i L_i$$ for n lights



## High-Resolution Reflectance Field

$$C(x,y) = \int_{\Omega_h} W_h(\xi)T(\xi)d\xi + \int_{\Omega_l} W_l(\omega_i)L(\omega_i)d\omega$$

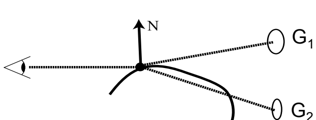
- Use techniques of environment matting [Chuang et al., SIGGRAPH 00].



## High-Resolution Reflectance Field

- Approximate $W_h$ by a sum of up to two Gaussians:
  - Reflective $G_1$.
  - Refractive $G_2$.

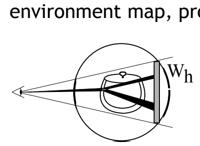
$$W_h(\xi) = a_1G_1 + a_2G_2$$





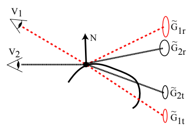

## Reproject □$_h$

- Project environment mattes onto the new environment.
  - Environment mattes acquired was parameterized on plane T (the plasma display).
  - We need to project the Gaussians to the new environment map, producing new Gaussians.




## View Interpolation

- Render low-resolution reflectance field.
- High-resolution reflectance field:
  - Match reflected and refracted Gaussians.




  - Interpolate *direction vectors*, not colors.
  - Determine new color along interpolated direction.



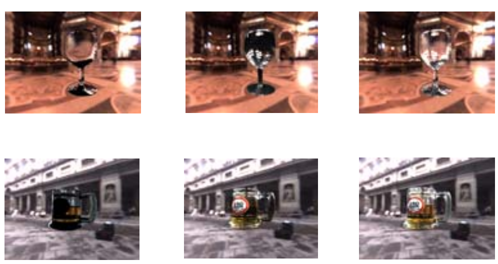
## Results

- Performance for 6x72 = 432 viewpoints
- 337,824 images taken in total !!
  - Acquisition (47 hours)
    - Alpha mattes – 1 hour
    - Environment mattes – 18 hours
    - Reflectance images – 28 hours
  - Processing
    - Opacity hull ~ 30 minutes
    - PCA Compression ~ 20 hours (MATLAB, unoptimized)
  - Rendering ~ 5 minutes per frame
  - Size
    - Opacity hull ~ 30 - 50 MB
    - Environment mattes ~ 0.5 - 2 GB
    - Reflectance images ~ Raw 370 GB / Compressed 2 - 4 GB



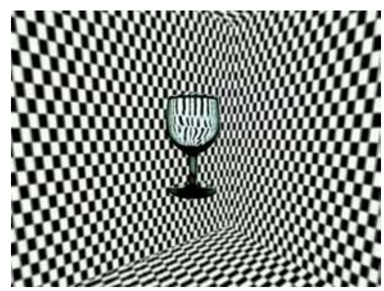
## Results

High-resolution $\Omega_h$ Low-resolution $\Omega_l$ Combined





## Results





## Results



## Results – $\Omega_h$

## Results – $\Omega_l$



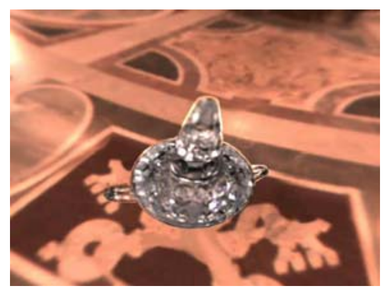
## Results – Combined



## Results



## Results





## Conclusions

- Data driven modeling is able to capture and render any type of object.
- Opacity hulls provide realistic 3D graphics models.
- Our models can be seamlessly inserted into new environments.
- Point-based rendering offers high image-quality for display of acquired models.



## Future Directions

- Real-time rendering
  - Done! [Vlasic et al., I3D 2003]
- Better environment matting
  - More than two Gaussians
- Better compression
  - MPEG-4 / JPEG 2000

## Acknowledgements

- Colleagues:
  - MIT: Chris Buehler, Tom Buehler
  - MERL: Bill Yerazunis, Darren Leigh, Michael Stern
- Thanks to:
  - David Tames, Jennifer Roderick Pfister
- NSF grants CCR-9975859 and EIA-9802220
- Papers available at:
  http://www.merl.com/people/pfister/

## Eurographics 2003

# Point-Based Computer Graphics

Marc Alexa, Carsten Dachsbacher,
Markus Gross, Mark Pauly,
Hanspeter Pfister, Marc Stamminger,
Matthias Zwicker

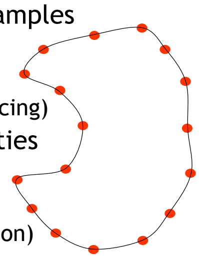
## Point-based Surface Reps

- Marc Alexa
- Discrete Geometric Modeling Group
- Darmstadt University of Technology
- alexa@informatik.tu-darmstadt.de



## Motivation

- Many applications need a definition of surface based on point samples
  - Reduction
  - Up-sampling
  - Interrogation (e.g. ray tracing)
- Desirable surface properties
  - Manifold
  - Smooth
  - Local (efficient computation)





## Overview

- Introduction & Basics
- Fitting Implicit Surfaces
- Projection-based Surfaces



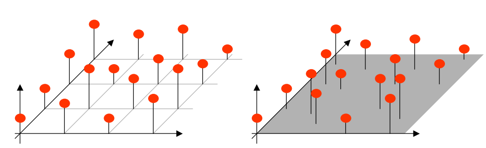
## Introduction & Basics

- Terms
  - Regular/Irregular, Approximation/Interpolation, Global/Local
- Standard interpolation/approximation techniques
  - Triangulation, Voronoi-Interpolation, Least Squares (LS), Radial Basis Functions (RBF), Moving LS
- Problems
  - Sharp edges, feature size/noise
- Functional -> Manifold



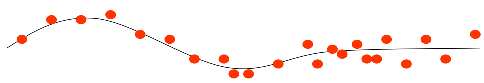
## Terms: Regular/Irregular

- Regular (on a grid) or irregular (scattered)
- Neighborhood (topology) is unclear for irregular data

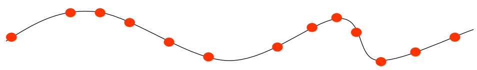


## Terms: Approximation/Interpolation

- Noisy data -> Approximation



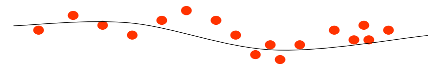
- Perfect data -> Interpolation



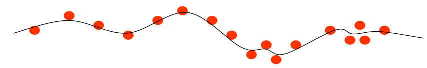


## Terms: Global/Local

- Global approximation



- Local approximation

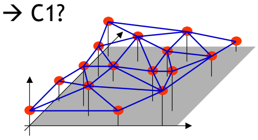


- Locality comes at the expense of smoothness



## Triangulation

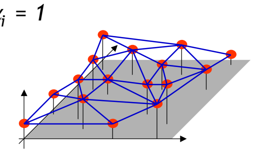
- Exploit the topology in a triangulation (e.g. Delaunay) of the data
- Interpolate the data points on the triangles
  - Piecewise linear → C0
  - Piecewise quadratic → C1?
  - …





## Triangulation: Piecewise linear

- Barycentric interpolation on simplices (triangles)
  - given $d+1$ points $x_i$ with values $f_i$ and a point $x$ inside the simplex defined by $x_i$
  - Compute $\alpha_i$ from $x = \Sigma_i\, \alpha_i \cdot x_i$ and $\Sigma_i\, \alpha_i = 1$
  - Then $f = \Sigma_i\, \alpha_i \cdot f_i$





## Voronoi Interpolation

- compute Voronoi diagram
- for any point $x$ in space
  - add $x$ to Voronoi diagram
  - Voronoi cell $\tau$ around $x$ intersects original cells $\tau_i$ of natural neighbors $n_i$
  - interpolate

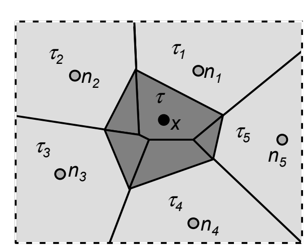
$$f(x) = \frac{\sum_i \lambda_i(x) \cdot \left(f_i + \nabla f_i^T \cdot (x - x_i)\right)}{\sum_i \lambda_i(x)}$$

with

$$\lambda_i(x) = \frac{|\tau \cap \tau_i|}{|\tau| \cdot \|x - x_i\|}$$



## Voronoi Interpolation



## Voronoi Interpolation

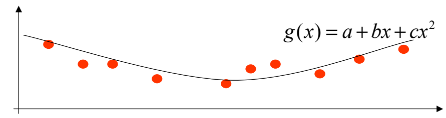
***Properties of Voronoi Interpolation:***

- linear Precision
- local
- for $d = 1 \rightarrow f(x)$ piecewise cubic
- $f(x) \in C^1$ on domain
- $f(x, x_1, \ldots, x_n)$ is continuous in $x_i$



## Least Squares

- Fits a primitive to the data
- Minimizes squared distances between the $p_i$'s and primitive $g$




$$\min_g \sum_i \left(p_{i_y} - g(p_{i_x})\right)^2$$



## Least Squares - Example

- Primitive is a polynomial

$$g(x) = (1, x, x^2, \ldots) \cdot \mathbf{c}^T$$

- $\min \sum_i \left(p_{i_y} - (1, p_{i_x}, p_{i_x}^2, \ldots)\mathbf{c}^T\right)^2 \Rightarrow$

$$0 = \sum_i 2 p_{i_x}^j \left(p_{i_y} - (1, p_{i_x}, p_{i_x}^2, \ldots)\mathbf{c}^T\right)$$

- Linear system of equations



## Least Squares - Example

- Resulting system

$$0 = \sum_i 2 p_{i_x}^j \left(p_{i_y} - (1, p_{i_x}, p_{i_x}^2, \ldots)\mathbf{c}^T\right) \Leftrightarrow$$

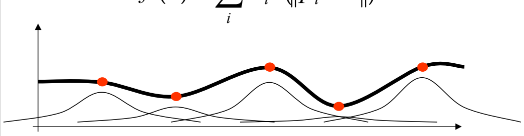
$$\begin{pmatrix} 1 & x & x^2 & \cdots \\ x & x^2 & x^3 & \\ x^2 & x^3 & x^4 & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} y \\ yx \\ yx^2 \\ \vdots \end{pmatrix}$$



## Radial Basis Functions

- Represent interpolant as
  - Sum of radial functions $r$
  - Centered at the data points $p_i$

$$f(x) = \sum_i w_i r(\|p_i - x\|)$$





## Radial Basis Functions

- Solve $p_{j_y} = \sum_i w_i r(\|p_{i_x} - p_{j_x}\|)$

  to compute weights $w_i$
- Linear system of equations

$$\begin{pmatrix} r(0) & r(\|p_{0_x} - p_{1_x}\|) & r(\|p_{0_x} - p_{2_x}\|) & \cdots \\ r(\|p_{1_x} - p_{0_x}\|) & r(0) & r(\|p_{1_x} - p_{2_x}\|) & \\ r(\|p_{2_x} - p_{0_x}\|) & r(\|p_{2_x} - p_{1_x}\|) & r(0) & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} p_{0_y} \\ p_{1_y} \\ p_{2_y} \\ \vdots \end{pmatrix}$$

## Radial Basis Functions

- Solvability depends on radial function
- Several choices assure solvability
  - $r(d) = d^2 \log d$ (thin plate spline)
  - $r(d) = e^{-d^2/h^2}$ (Gaussian)
    - $h$ is a data parameter
    - $h$ reflects the feature size or anticipated spacing among points



## Function Spaces!

- Monomial, Lagrange, RBF share the same principle:
  - Choose basis of a function space
  - Find weight vector for base elements by solving linear system defined by data points
  - Compute values as linear combinations
- Properties
  - One costly preprocessing step
  - Simple evaluation of function in any point



## Function Spaces?

- Problems
  - Many points lead to large linear systems
  - Evaluation requires global solutions
- Solutions
  - RBF with compact support
    - Matrix is sparse
    - Still: solution depends on every data point, though drop-off is exponential with distance
  - Local approximation approaches



## Shepard Interpolation

- Approach for $\mathbf{R}^d$: $f(x) = \Sigma_i\, \phi_i(x)\, f_i$

  with basis functions $\phi_i(x) = \dfrac{\|x - x_i\|^{-p}}{\sum_j \|x - x_j\|^{-p}}$

- define $f(x_i) := f_i = \lim_{x \to x_i} f(x)$



## Shepard Interpolation

- $f(x)$ is a convex combination of $\phi_i$, because all $\phi_i(\mathbf{R}^d) \subseteq [0,1]$ and $\Sigma_i\, \phi_i(x) \equiv 1$.
  - ➔ $f(x)$ is contained in the convex hull of data points
- for $p>1$ $f(p) \in \mathbf{C}^\infty$ and $\nabla_x \phi_i(x_i) = 0$
  - ➔ Data points are saddles
- global interpolation
  - ➔ every $f(x)$ depends on all data points
- Only constant precision, i.e. only constant functions are reproduced exactly



## Shepard Interpolation

Localization:

- Set $f(x) = \Sigma_i\, \mu_i(x) \cdot \phi_i(x) \cdot f_i$

  with $\mu_i(x) = \begin{cases} \left(1 - \frac{\|x - x_i\|}{R_i}\right)^{\nu} & \text{für } \|x - x_i\| < R_i \\ 0 & \text{sonst} \end{cases}$

  for reasonable $R_i$ and $\nu > 1$

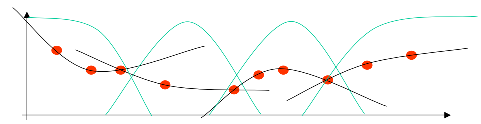
  ➔ no constant precision because of possible holes in the data

## Spatial subdivisions

- Subdivide parameter domain into overlapping cells $\tau_i$ with centroids $c_i$
- Compute Shepard weights $\phi_i(x) = \frac{\|x - c_i\|^{-p}}{\sum_j \|x - c_j\|^{-p}}$ and localize them using the radius of the cell
- Interpolate/approximate data points in each cell by an arbitrary function $f_i$
- The interpolant is given as $f(x) = \Sigma_i\ \mu_i(x) \cdot \phi_i(x) \cdot f_i$



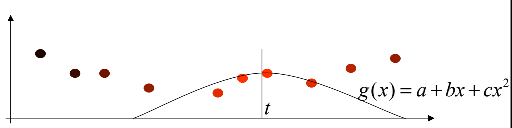
## Spatial subdivisions





## Moving Least Squares

- Compute a local LS approximation at $t$
- Weight data points based on distance to $t$




$$\min \sum_i \left(p_{i_y} - g\left(p_{i_x}\right)\right)^2 \theta\left(\left\|t - p_{i_x}\right\|\right)$$

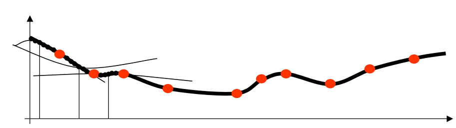


## Moving Least Squares

- The set

$$f(t) = g_t(t), g_t : \min_g \sum_i \left(p_{i_y} - g\left(p_{i_x}\right)\right)^2 \theta\left(\left\|t - p_{i_x}\right\|\right)$$

is a smooth curve, iff $\theta$ is smooth





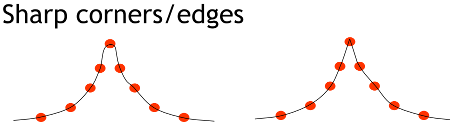
## Moving Least Squares

- Typical choices for $\theta$:
  - $\theta(d) = d^{-r}$
  - $\theta(d) = e^{-d^2/h^2}$
- Note: $\theta_i = \theta\left(\left\|t - p_{i_x}\right\|\right)$ is fixed
- For each $t$
  - Standard weighted LS problem
  - Linear iff corresponding LS is linear



## Typical Problems

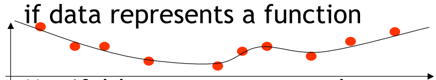
- Sharp corners/edges



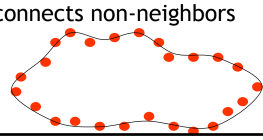
- Noise vs. feature size

## Functional -> Manifold

- Standard techniques are applicable if data represents a function



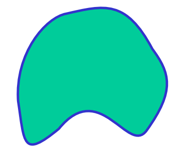
- Manifolds are more general
  - No parameter domain
  - No knowledge about neighbors, Delaunay triangulation connects non-neighbors





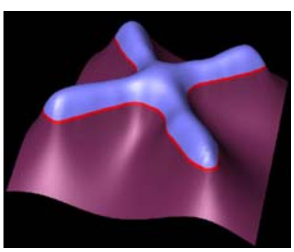
## Implicits

- Each orientable n-manifold can be embedded in n+1 - space
- Idea: Represent n-manifold as zero-set of a scalar function in n+1 - space
  - Inside: $f(\mathbf{x}) < 0$
  - On the manifold: $f(\mathbf{x}) = 0$
  - Outside: $f(\mathbf{x}) > 0$





## Implicits - Illustration

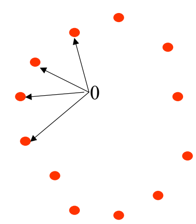


- Image courtesy Greg Turk



## Implicits from point samples

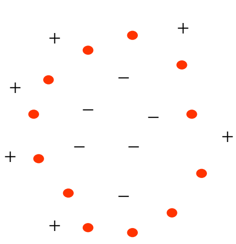
- Function should be zero in data points
  - $f(\mathbf{p}_i) = 0$
- Use standard approximation techniques to find $f$
- Trivial solution: $f = 0$
- Additional constraints are needed






## Implicits from point samples

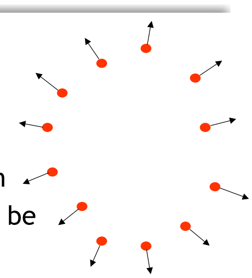
- Constraints define inside and outside
- Simple approach (Turk, O'Brien)
  - Sprinkle additional information manually
  - Make additional information soft constraints





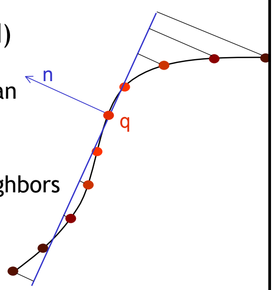
## Implicits from point samples

- Use normal information
- Normals could be computed from scan
- Or, normals have to be estimated



## Estimating normals

- Normal orientation (Implicits are signed)
  - Use inside/outside information from scan
- Normal direction by fitting a tangent
  - LS fit to nearest neighbors
  - Weighted LS fit
  - MLS fit



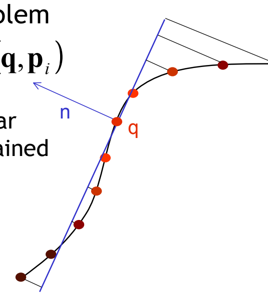



## Estimating normals

- General fitting problem

$$\min_{\|\mathbf{n}\|=1} \sum_i \langle \mathbf{q}-\mathbf{p}_i, \mathbf{n} \rangle^2 \theta(\mathbf{q}, \mathbf{p}_i)$$

  - Problem is non-linear because n is constrained to unit sphere






## Estimating normals

- The constrained minimization problem

$$\min_{\|\mathbf{n}\|=1} \sum_i \langle \mathbf{q}-\mathbf{p}_i, \mathbf{n} \rangle^2 \theta_i$$

is solved by the eigenvector corresponding to the smallest eigenvalue of

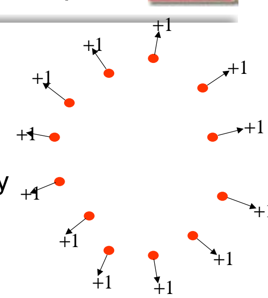
$$\begin{pmatrix} \sum_i (q_x - p_{i_x})^2 \theta_i & \sum_i (q_x - p_{i_y})^2 \theta_i & \sum_i (q_x - p_{i_z})^2 \theta_i \\ \sum_i (q_y - p_{i_x})^2 \theta_i & \sum_i (q_y - p_{i_y})^2 \theta_i & \sum_i (q_y - p_{i_z})^2 \theta_i \\ \sum_i (q_z - p_{i_x})^2 \theta_i & \sum_i (q_z - p_{i_y})^2 \theta_i & \sum_i (q_z - p_{i_z})^2 \theta_i \end{pmatrix}$$



## Implicits from point samples

- Compute non-zero anchors in the distance field
- Use normal information directly as constraints

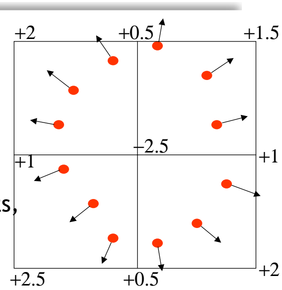
$$f(\mathbf{p}_i + \mathbf{n}_i) = 1$$






## Implicits from point samples

- Compute non-zero anchors in the distance field
- Compute distances at specific points
  - Vertices, mid-points, etc. in a spatial subdivision






## Computing Implicits

- Given N points and normals $p_i, n_i$ and constraints $f(\mathbf{p}_i) = 0, f(\mathbf{c}_i) = d_i$
- Let $\mathbf{p}_{i+N} = \mathbf{c}_i$
- An RBF approximation

$$f(\mathbf{x}) = \sum_i w_i r(\|\mathbf{p}_i - \mathbf{x}\|)$$

leads to a system of linear equations

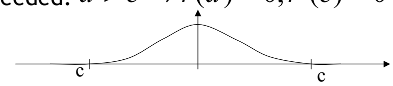
## Computing Implicits

- Practical problems: $N > 10000$
- Matrix solution becomes difficult
- Two solutions
  - Sparse matrices allow iterative solution
  - Smaller number of RBFs



## Computing Implicits

- Sparse matrices $\begin{pmatrix} r(0) & r(\|p_0-p_1\|) & r(\|p_0-p_2\|) & \Lambda \\ r(\|p_1-p_0\|) & r(0) & r(\|p_1-p_2\|) & \\ r(\|p-p_0\|) & r(\|p_2-p_1\|) & r(0) & \\ M & & & O \end{pmatrix}$
  - Needed: $d > c \rightarrow r(d) = 0, r'(c) = 0$

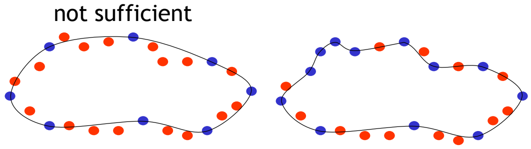



  - Compactly supported RBFs



## Computing Implicits

- Smaller number of RBFs
- Greedy approach (Carr et al.)
  - Start with random small subset
  - Add RBFs where approximation quality is not sufficient





## RBF Implicits - Results

- Images courtesy Greg Turk



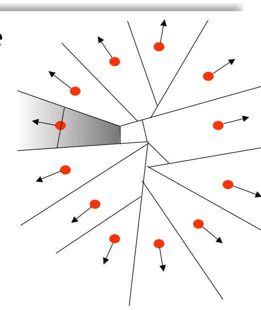

## RBF Implicits - Results

- Images courtesy Greg Turk



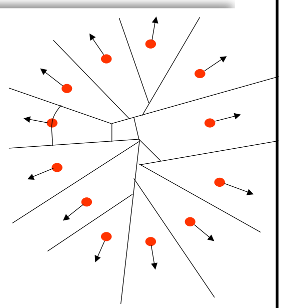

## Hoppe's approach

- Use linear distance field per point
  - Direction is defined by normal
- In every point in space use the distance field of the closest point



## Hoppe's approach - smoother

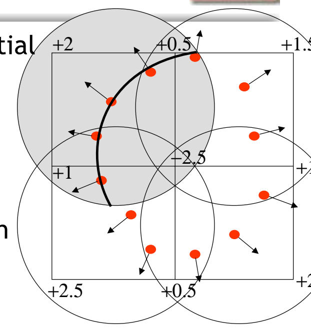
- Direction fields are interpolated using Voronoi interpolation





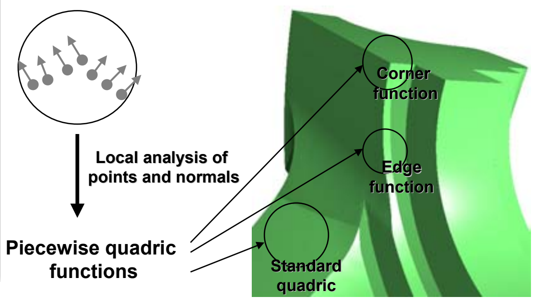
## PuO Implicits

- Construct a spatial subdivision
- Compute local distance field approximations
  - e.g. Quadrics
- Blend them with local Shepard weights





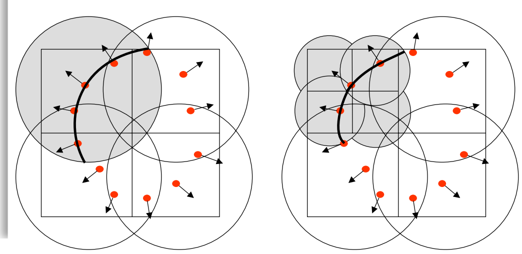

## PuO Implicits: Sharp features






## Multi-level PuO Implicits

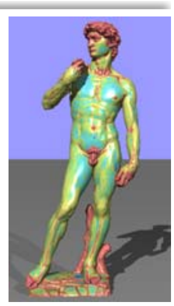
- Subdivide cells based on local error





## Multi-level PuO Implicits

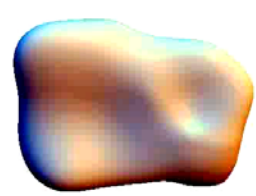
- Local computations
  - Insensitive to number of points
- Local adaptation to shape complexity
- Sensitive to output complexity





## Multi-level PuO Implicits

- Aproximation at arbitrary accuracy



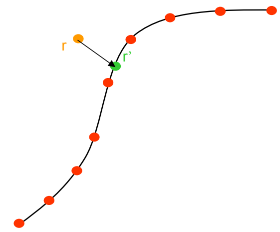
## Implicits - Conclusions

- Scalar field is underconstrained
  - Constraints only define where the field is zero, not where it is non-zero
  - Additional constraints are needed
- Signed fields restrict surfaces to be unbounded
  - All implicit surfaces define solids



## Projection

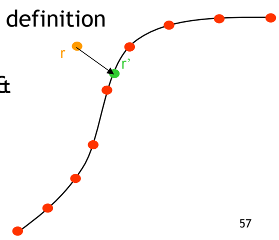
- Idea: Map space to surface
- Surface is defined as fixpoints of mapping






## Surface definition

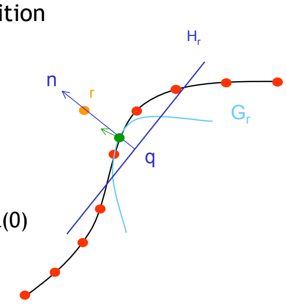
- Projection procedure (Levin)
  - Local polyonmial approximation
    - Inspired by differential geometry
  - "Implicit" surface definition
  - Infinitely smooth &
  - Manifold surface






## Surface Definition

- Constructive definition
  - Input point r
  - Compute a local reference plane $H_r$=<q,n>
  - Compute a local polynomial over the plane $G_r$
  - Project point $r'=G_r(0)$
  - Estimate normal

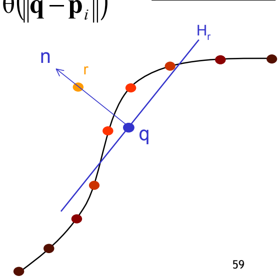





## Local Reference Plane

- Find plane $H_r = \langle \mathbf{q}, \mathbf{n} \rangle + D$
  - $\min_{\mathbf{q}, \|\mathbf{n}\|=1} \sum_i \langle \mathbf{q} - \mathbf{p}_i, \mathbf{n} \rangle^2 \theta(\|\mathbf{q} - \mathbf{p}_i\|)$
  - $\theta(\mathrm{d}) = e^{d^2/h^2}$
    - $h$ is feature size/ point spacing
  - $H_r$ is independent of $r$'s distance
  - Manifold property

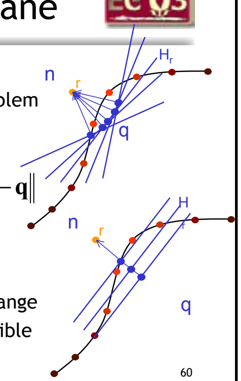
Weight function based on distance to q, not r





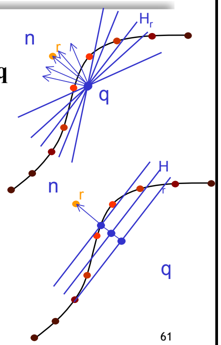

## Local Reference Plane

- Computing reference plane
  - Non-linear optimization problem
- Minimize independent variables:
  - Over $\mathbf{n}$ for fixed distance $\|\mathbf{r} - \mathbf{q}\|$
  - Along $\mathbf{n}$ for fixed direction $\mathbf{n}$
  - $\mathbf{q}$ changes -> the weights change
  - Only iterative solutions possible



## Local Reference Plane

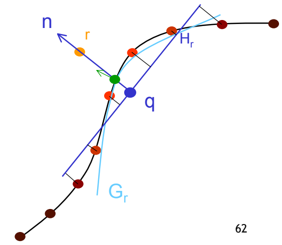
- Practical computation
  - Minimize over $\mathbf{n}$ for fixed $\mathbf{q}$
    - Eigenvalue problem
  - Translate q so that
    $$\mathbf{r} = \mathbf{q} + \|\mathbf{r} - \mathbf{q}\|\mathbf{n}$$
    - Effectively changes $\|\mathbf{r} - \mathbf{q}\|$
  - Minimize along $\mathbf{n}$ for fixed direction $\mathbf{n}$
    - Exploit partial derivative






## Projecting the Point

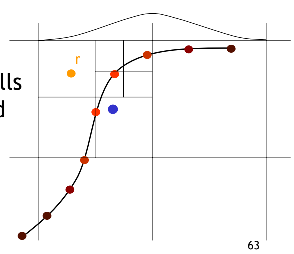
- MLS polyonomial over $H_r$
  - $\min_{G\in\Pi_d}\sum_i \left(\langle \mathbf{q}-\mathbf{p}_i, \mathbf{n}\rangle - G\left(\mathbf{p}_i|_{H_r}\right)\right)^2 \theta\left(\|\mathbf{q}-\mathbf{p}_i\|\right)$
  - LS problem
  - $r' = G_r(0)$
  - Estimate normal






## Spatial data structure

- Regular grid based on support of $\theta$
  - Each point influences only 8 cells
- Each cell is an octree
  - Distant octree cells are approximated by one point in center of mass






## Conclusions

- Projection-based surface definition
  - Surface is smooth and manifold
  - Surface may be bounded
  - Representation error mainly depends on point density
  - Adjustable feature size h allows to smooth out noise

Eurographics 2003

# Point-Based Rendering

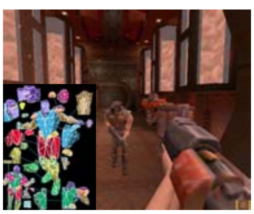
Matthias Zwicker
Computer Graphics Lab
ETH Zürich

## Point-Based Rendering

- Introduction and motivation
- Surface elements
- Rendering
- Antialiasing
- Hardware Acceleration
- Conclusions



## Motivation 1



Quake 2, 1998
10k triangles

Nvidia, 2002
millions of triangles



## Motivation 1

- Performance of 3D hardware has exploded (e.g., GeForce4: 136 million vertices per second)
- Projected triangles are very small (i.e., cover only a few pixels)
- Overhead for triangle setup increases (initialization of texture filtering, rasterization)

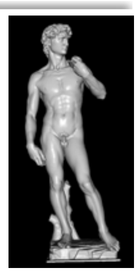
➡ A simpler, more efficient rendering primitive than triangles?



## Motivation 2

- Modern 3D scanning devices (e.g., laser range scanners) acquire huge point clouds
- Generating consistent triangle meshes is time consuming and difficult

➡ A rendering primitive for direct visualization of point clouds, without the need to generate triangle meshes?

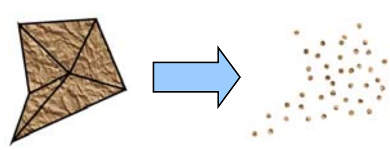


4 million pts.
[Levoy et al. 2000]



## Points as Rendering Primitives

- Point clouds instead of triangle meshes [Levoy and Whitted 1985]
- 2D vector versus pixel graphics



triangle mesh (with textures)

point cloud

## Point-Based Surface Representation

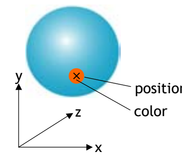
- Points are ***samples*** of the surface
- The point cloud describes:
  - 3D geometry of the surface
  - Surface reflectance properties (e.g., diffuse color, etc.)
- There is no additional information, such as
  - connectivity (i.e., explicit neighborhood information between points)
  - texture maps, bump maps, etc.



## Surface Elements - Surfels

- Each point corresponds to a surface element, or ***surfel***, describing the surface in a small neighborhood
- Basic surfels:

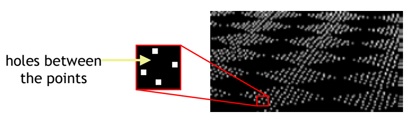
```
BasicSurfel {
  position;
  color;
}
```






## Surfels

- How to represent the surface between the points?



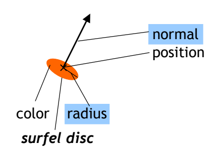

- Surfels need to ***interpolate*** the surface between the points
- A certain ***surface area*** is associated with each surfel



## Surfels

- Surfels can be extended by storing additional attributes
- This allows for higher quality rendering or advanced shading effects

```
ExtendedSurfel {
  position;
  color;
  normal;
  radius;
  etc...
}
```





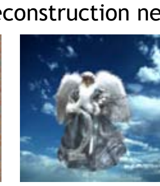

## Surfels

- Surfels store essential information for ***rendering***
- Surfels are primarily designed as a ***point rendering primitive***
- They do not provide a mathematically smooth surface definition (see [Alexa 2001], point set surfaces)



## Model Acquisition

- 3D scanning of physical objects
  - See Pfister, acquisition
  - Direct rendering of acquired point clouds
  - No mesh reconstruction necessary

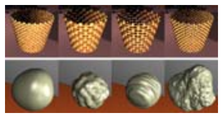


[Matusik et al. 2002]

## Model Acquisition

- Sampling synthetic objects
  - Efficient rendering of complex models
  - Dynamic sampling of procedural objects and animated scenes (see Stamminger, dynamic sampling)

[Zwicker et al. 2001]

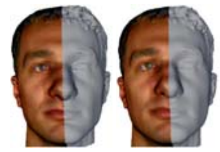


[Stamminger et al. 2001]



## Model Acquisition

- Processing and editing of point-sampled geometry



spectral processing
[Pauly, Gross 2002]
(see Gross, spectral processing)

point-based surface editing
[Zwicker et al. 2002]
(see Pauly, Pointshop3D)



## Point Rendering Pipeline

Point Cloud

Projection → Shading → Visibility → Image Reconstruction

Frame-buffer

- Simple, pure forward mapping pipeline
- Surfels carry all information through the pipeline („surfel stream")
- No texture look-ups
- Framebuffer stores RGB, alpha, and Z



## Point Rendering Pipeline

**Projection** → **Shading** → **Visibility** → **Image Reconstruction**

- Perspective projection of each point in the point cloud
- Analogous to projection of triangle vertices
  - homogeneous matrix-vector product
  - perspective division



## Point Rendering Pipeline

**Projection** → **Shading** → **Visibility** → **Image Reconstruction**

- Per-point shading
- Conventional models for shading (Phong, Torrance-Sparrow, reflections, etc.)

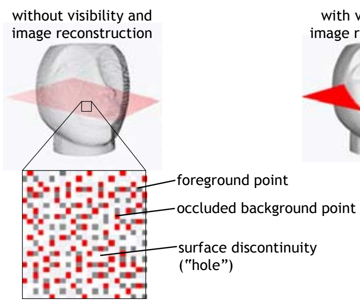


## Point Rendering Pipeline

**Projection** → **Shading** → **Visibility** → **Image Reconstruction**

- Visibility and image reconstruction is tightly coupled
  - Discard points that are occluded from the current viewpoint
  - Reconstruct continuous surfaces from projected points (antialiasing)

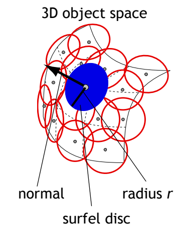
## Visibility and Image Reconstruction






## Visibility and Image Reconstruction

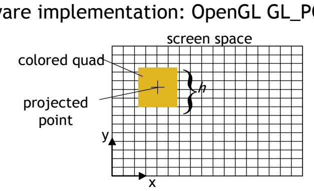
- Goal: avoid holes and discard occluded surfels
- Use surfel discs with radius *r* to cover surface completely
- Apply z-buffer to discard invisible surfels






## Quad Rendering Primitive

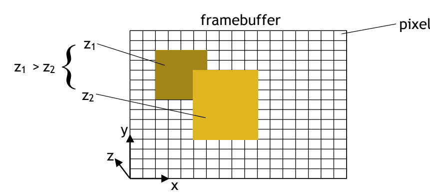
- Rasterize a colored quad centered at the projected point, use z-buffering
- The quad side length is *h*, where *h* = 2 * *r* * *s*
- The scaling factor *s* given by perspective projection and viewport transformation
- Hardware implementation: OpenGL GL_POINTS






## Visibility: Z-Buffering

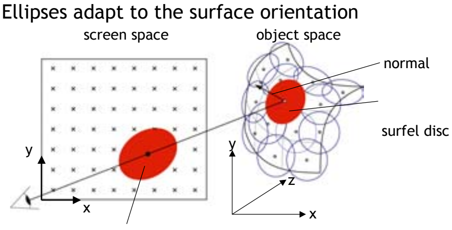
- ***No blending*** of rendering primitives






## Projected Disc Rendering Primitive

- Project surfel discs from object to screen space
- Projecting discs results in ellipses in screen space
- Ellipses adapt to the surface orientation



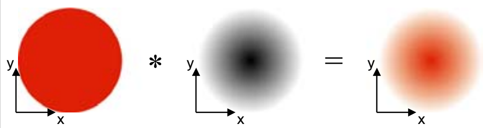



## Discussion

- Quad and projected disc primitive
  - Simple, efficient
  - Hardware support
  - Low image quality
  - Suitable for preview renderers (e.g. Qsplat [Rusinkiewicz et al. 2000] )
- Problem: no blending of primitives

## Splatting

- A splat primitive consists of a colored point primitive and an alpha mask




colored point primitive *c*

alpha mask *w(x,y)* (often a 2D Gauss function)

splat primitive *c * w(x,y)*



## Splatting

- The final color *c(x,y)* is computed by ***additive alpha blending***, i.e., by computing the weighted sum

color of splat *i* — alpha of splat *i* at position *(x,y)*

$$c(x,y) = \frac{\sum_i c_i w_i(x,y)}{\sum_i w_i(x,y)}$$

- Normalization is necessary, because the weights do not sum up to one with irregular point distributions

$$\sum_i w_i(x,y) \neq 1$$

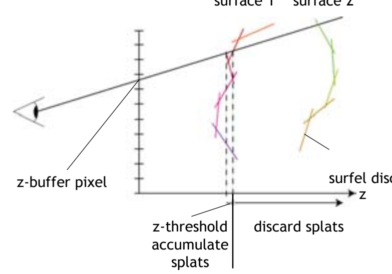


## Splatting




## Splatting

- Extended z-buffering





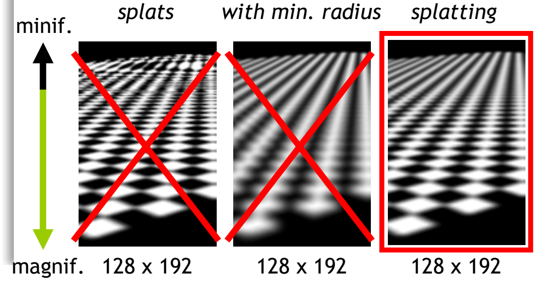

## Extended Z-Buffering

```
DepthTest(x,y) {
  if (abs(splat z – z(x,y)) < threshold) {
    c(x,y) = c(x,y) + splat color
    w(x,y) = w(x,y) + splat w(x,y)
  } else if (splat z < z(x,y)) {
    z(x,y) = splat z
    c(x,y) = splat color
    w(x,y) = splat w(x,y)
  }
}
```



## Splatting Comparison

*elliptical splats* | *circular splats with min. radius* | *surface splatting*



# High Quality Splatting

- High quality splatting requires careful analysis of ***aliasing*** issues
  - Review of signal processing theory
  - Application to point rendering
  - Surface splatting [Zwicker et al. 2001]



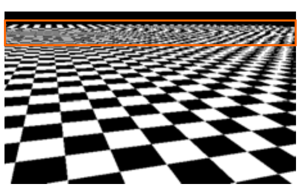
# Aliasing in Computer Graphics

- Aliasing = Sampling of continuous functions below the ***Nyquist frequency***
  - To avoid aliasing, sampling rate must be twice as high as the maximum frequency in the signal
- Aliasing effects:
  - Loss of detail
  - Moire patterns, jagged edges
  - Disintegration of objects or patterns
- Aliasing in Computer Graphics
  - Texture Mapping
  - Scan conversion of geometry



# Aliasing in Computer Graphics

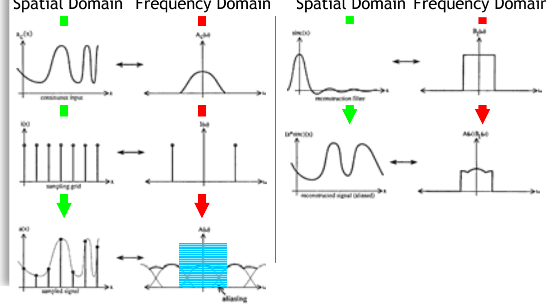
- Aliasing: high frequencies in the input signal appear as low frequencies in the reconstructed signal





# Occurrence of Aliasing

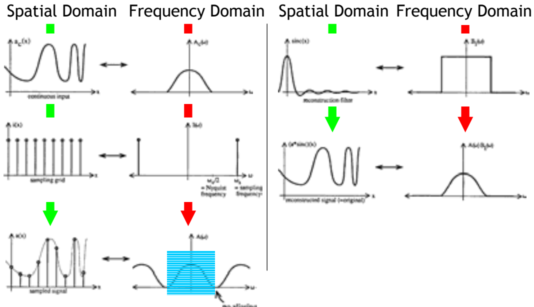
Spatial Domain Frequency Domain Spatial Domain Frequency Domain






# Aliasing-Free Reconstruction

Spatial Domain Frequency Domain Spatial Domain Frequency Domain

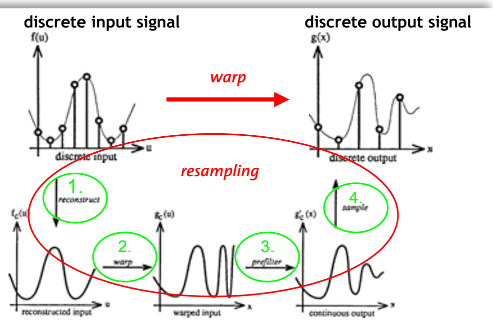





# Antialiasing

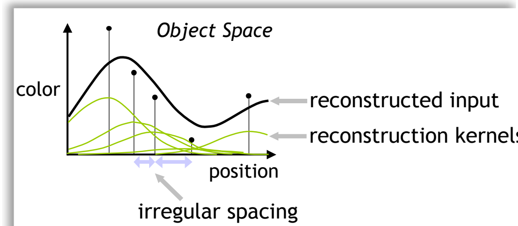
- Prefiltering
  - Band-limit the continuous signal before sampling
  - Eliminates all aliasing (with an ideal low-pass filter)
  - Closed form solution not available in general
- Supersampling
  - Raise sampling rate
  - Reduces, but does not eliminate all aliasing artifacts (in practice, many signals have infinite frequencies)
  - Simple implementation (hardware)

# Resampling



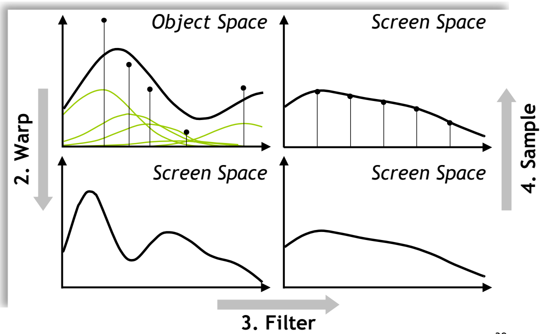



# Resampling Filters



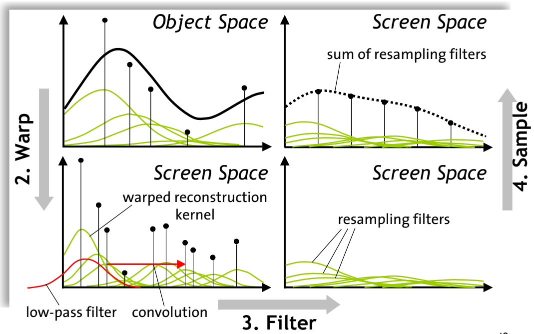



# Resampling Filters






# Resampling Filters





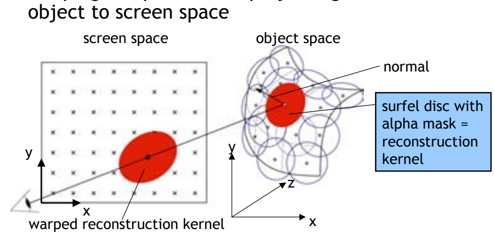

# Resampling

- Resampling in the context of surface rendering
  - Discrete input function = surface texture (discrete 2D function)
  - Warping = projecting surfaces to the image plane (2D to 2D projective mapping)



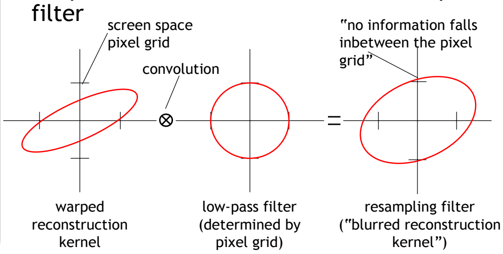
# 2D Reconstruction Kernels

- 2D reconstruction kernels are given by surfel discs with alpha masks
- Warping is equivalent to projecting the kernel from object to screen space



## Resampling Filters

- A resampling filter is a convolution of a warped reconstruction filter and a low-pass filter






## Mathematical Formulation

resampling filter

$$c(x,y)=\sum_k c_k r_k(m^{-1}(x,y))\otimes h(x,y)$$

pixel color

warping function

low pass filter

reconstruction kernel

reconstruction kernel color



## Gaussian Resampling Filters

- Gaussians are closed under linear warping and convolution
- With Gaussian reconstruction kernels and low-pass filters, the resampling filter is a Gaussian, too
- Efficient rendering algorithms (***surface splatting*** [Zwicker et al. 2001])

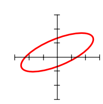


## Mathematical Formulation

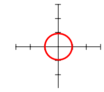
$$c(x,y)=\sum_k c_k r_k(m^{-1}(x,y))\otimes h(x,y)$$

Gaussian reconstruction kernel

Gaussian low-pass filter



*screen space*



*screen space*



## Mathematical Formulation

$$c(x,y)=\sum_k c_k r_k(m^{-1}(x,y))\otimes h(x,y)$$

$$=\sum_k c_k G_k(x,y)$$

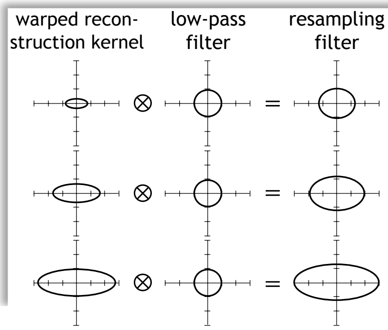
Gaussian resampling filter



## Algorithm

```
for each point P {
  project P to screen space;
  shade P;
  determine resampling kernel G;
  splat G;
}
for each pixel {
  normalize;
}
```

## Properties of 2D Resampling Filters



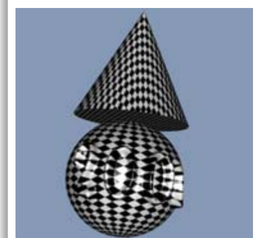

minification

magnification

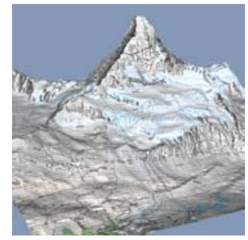


## Results

- High quality reconstruction and filtering



200k points



4783k points

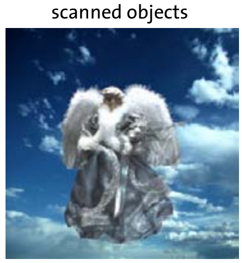


## Results

transparent surfaces

987k points

scanned objects



[MERL/MIT Matusik et al.]



## Hardware Implementation

- Based on the object space formulation of EWA filtering
- Implemented using textured triangles
- All calculations are performed in the programmable hardware (extensive use of vertex shaders)
- Presented at EG 2002 ([Ren et al. 2002])



## Surface Splatting Performance

- Software implementation
  - 500 000 splats/sec on 866 MHz PIII
  - 1 000 000 splats/sec on 2 GHz P4
- Hardware implementation [Ren et al. 2002]
  - Uses texture mapping and vertex shaders
  - 3 000 000 splats/sec on GeForce4 Ti 4400



## Conclusions

- Points are an efficient rendering primitive for highly complex surfaces
- Points allow the direct visualization of real world data acquired with 3D scanning devices
- High performance, low quality point rendering is supported by 3D hardware (tens of millions points per second)
- High quality point rendering with anisotropic texture filtering is available
  - 3 million points per second with hardware support
  - 1 million points per second in software
- Antialiasing technique has been extended to volume rendering

## Applications

- Direct visualization of point clouds
- Real-time 3D reconstruction and rendering for virtual reality applications
- Hybrid point and polygon rendering systems
- Rendering animated scenes
- Interactive display of huge meshes
- On the fly sampling and rendering of procedural objects



## Future Work

- Dedicated rendering hardware
- Efficient approximations of exact EWA splatting
- Rendering architecture for on the fly sampling and rendering



## Acknowledgments


- Hanspeter Pfister, Jeroen van Baar (MERL, Cambridge MA)
- Markus Gross, Mark Pauly, CGL
- Liu Ren

http://graphics.ethz.ch/surfels
http://graphics.ethz.ch/pointshop3d




## References


- [Levoy and Whitted 1985] The use of points as a display primitive, technical report, University of North Carolina at Chapel Hill, 1985
- [Heckbert 1986] Fundamentals of texture mapping and image warping, Master's Thesis, 1986
- [Grossman and Dally 1998] Point sample rendering, Eurographics workshop on rendering, 1998
- [Levoy et al. 2000] The digital Michelangelo project, SIGGRAPH 2000
- [Rusinkiewicz et al. 2000] Qsplat, SIGGRAPH 2000
- [Pfister et al. 2000] Surfels: Surface elements as rendering primitives, SIGGRAPH 2000
- [Zwicker et al. 2001] Surface splatting, SIGGRAPH 2001
- [Zwicker et al. 2002] EWA Splatting, to appear, IEEE TVCG 2002
- [Ren et al. 2002] Object space EWA splatting: A hardware accelerated approach to high quality point rendering, Eurographics 2002

**Eurographics 2003**

# Point-Based Computer Graphics

Marc Alexa, Carsten Dachsbacher,
Markus Gross, Mark Pauly,
Hanspeter Pfister, Marc Stamminger,
Matthias Zwicker

## Introduction

- point rendering
  - how adapt point densities?
    - *for a given viewing position, how can we get n points that suffice for that viewer?*
  - how render the points?
    - *given n points, how can we render an image from them ?*



## Introduction

- how render the points?
  - project point to pixel, set pixel color
  - hardware solution (Radeon 9700 Pro)
    - ~80 mio. points per second
    - no hole filling
  - software solution
    - ~8 mio. points per second
    - hole filling
- *hardware != software*



## Introduction

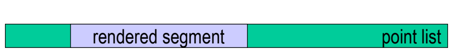
- even with hardware:
  - `for (int i = 0; i < N; i++) renderPointWithNormalAndColor (x[i],y[i],z[i],nx[i],ny[i],nz[i],…);`
    → 10 mio points per second
  - `for (int i = 0; i < N; i++) renderPoint(x[i],y[i],z[i]);`
    → 20 mio points per second
  - `float *p = {...} renderPoints(p);`
    → 80 mio points per second
- → *best performance with sequential processing of large chunks !*



## Introduction

- what we want:
  - sequential processing *and*
  - adaptive point densities

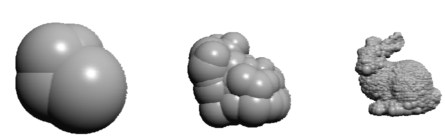
→ precomputed point lists
→ render continuous segments only






## Hierarchical Processing

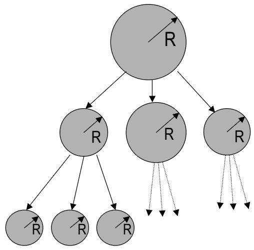
- Q-Splat
  - Rusinkiewicz et al., Siggraph 2000
  - hierarchical point rendering based on Bounding Sphere Hierarchy

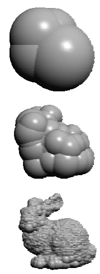


© S. Rusinkiewicz

## Hierarchical Processing

- Q-Splat hierarchy





## Hierarchical Processing

- Q-Splat recursive rendering

```
render( Node n ) {
  // compute screen size of node
  s = n.R / distanceToCamera( n );
  // screen size too big?
  if ( s > threshold )
    // → render children
    forall children c
      render( c );
  else
    // else draw node
    renderPoint( n.xyz );
}
```



## Hierarchical Processing

- not sequential
- no array, but tree structure
- most work on CPU
- CPU is bottleneck: ~8 mio points per second

→ sequential version ?



## Sequential Point Trees

- store with node $d_{min}$ = n.R / 1 Pixel

```
• render( Node n ) {
    // node too close?
    if ( distanceToCamera( n ) < n.dmin )
      // → render children
      forall children c
        render( c );
    else
      // else draw node
      renderPoint( n.xyz );
  }
```



## Sequential Point Trees

- node `n` is rendered if:
  - n is not too close and
  - parent is not rendered
- or
  - `distToCam( n ) < n.dmin`
  - `distToCam( n.parent ) ≥ n.parent.dmin`
- parent is too close, but node is far enough

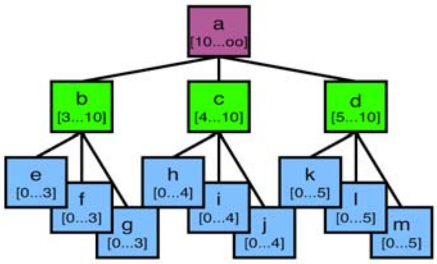


## Sequential Point Trees

- assume
  - `distToCam(n) ≈ distToCam(n.parent)`
- store with `n`
  - `n.dmax = n.parent.dmin`
- then a node is rendered if
  - `n.dmin ≤ distToCam(n) < n.dmax`

## Sequential Point Trees

- example tree






## Sequential Point Trees

- sequential version

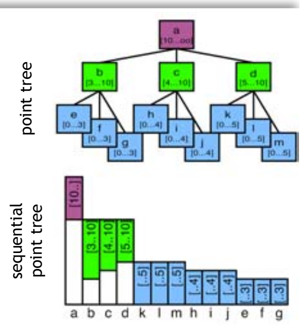
```
• foreach tree node n
    if ( n.dmin < distToCam(n) &&
         distToCam(n) < n.dmax )
       renderPoint(n);
```

- how enumerate nodes?



## Sequential Point Trees

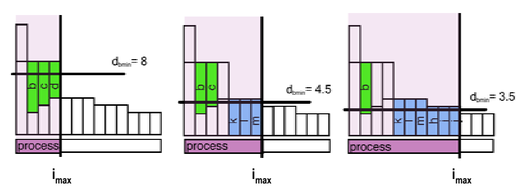
- sort nodes by $d_{max}$






## Sequential Point Trees

- compute lower bound $d_{bmin}$ on distToCam(n) with bounding volume
- all elements with $d_{max} < d_{bmin}$ can be skipped
- only prefix must be considered





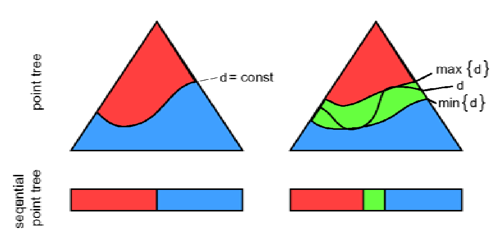

## Sequential Point Trees

- account for $d \neq d(\text{parent})$:
  - $d_{max} = d_{min}(\text{parent})$ + distance to parent
  - partially parent and some children selected
  - no visible artifacts from this



## Sequential Point Trees

- culling by GPU necessary, because d is not constant over object



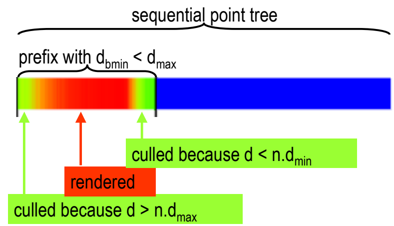
## Sequential Point Trees

- CPU does per frame:
  - compute $d_{bmin}$
  - search last node $i_{max}$ with $d_{max} > d_{bmin}$
  - send first $i_{max}$ points to GPU
- GPU then does for every node n
  - compute d = distToCam(n)
  - if $n.d_{min} \leq d \leq n.d_{max}$
    - render node



## Sequential Point Trees

- CPU does first interval selection by $d_{bmin}$
- GPU does fine granularity selection






## Sequential Point Trees

- Result
  - culling by GPU: only 10 - 40%
  - on a 2,4 GHz Pentium with Radeon 9700:
  - CPU-Load < 20% (usually much less)
  - > 50 Mio points *after* culling



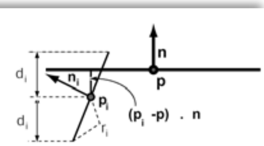
## Sequential Point Trees

- better error measurement
  - in flat regions
    - increase $d_{min}$, $d_{max}$
    - render larger points

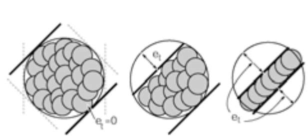


## Sequential Point Trees

- geometric
  - perpendicular error



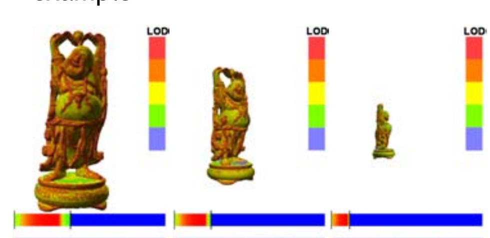

  - tangential error





## Sequential Point Trees

- example



## Sequential Point Trees

- also add texture criterion
- necessary for flat textured regions




## Sequential Point Trees

- if significant color variation in child nodes:
  - modify tangential error
  - increase error to node diameter
- prevents washed out colors in flat regions



## Sequential Point Trees

- perpendicular, tangential, texture error
- scale with 1/(view distance)
- fits into sequential point trees



## Sequential Point Trees

- combine errors
  - perpendicular $e_p$
  - tangential $e_t$
  - texture $e_{tex}$
- $e_{com} = \begin{cases} r & \text{if texture variation} \\ \sqrt{e_p^2 + e_t^2} & \text{else} \end{cases}$
- => screen error = $e_{com}$ / viewDistance



## Sequential Point Trees

- can be combined with polygons



## Sequential Point Trees

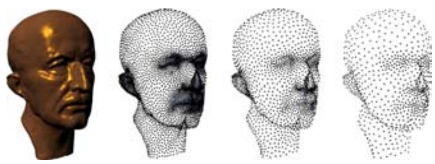
- combine with polygonal rendering
  - for every triangle
    - compute $d_{max}$ (longest side / $d_{max}$ = ε)
    - remove all points from triangle with smaller $d_{max}$
  - sort triangles for $d_{max}$
  - during rendering
    - for every object, compute upper bound $d_{bmax}$ on distance
    - send triangles with $d_{max} < d_{bmax}$ to GPU
    - on the GPU (vertex program)
      - test $d < d_{max}$
      - cull by alpha-test

## Sequential Point Trees

- pros
  - very simple!
  - CPU-load low
  - most work moved to GPU
  - GPU runs at maximum efficiency
- cons
  - no view frustum culling
  - currently: bad splatting support by GPU

## Eurographics 2003

# Efficient Simplification of Point-sampled Surfaces





## Overview

- Introduction
- Local surface analysis
- Simplification methods
- Error measurement
- Comparison



## Introduction

- Point-based models are often sampled very densely
- Many applications require coarser approximations, e.g. for efficient
  - Storage
  - Transmission
  - Processing
  - Rendering

⇨ We need simplification methods for reducing the complexity of point-based surfaces



## Introduction

- Example: Level-of-detail (LOD) rendering

10k 20k 60k 200k 2000k



## Introduction

- We transfer different simplification methods from triangle meshes to point clouds:
  - Hierarchical clustering
  - Iterative simplification
  - Particle simulation
- Depending on the intended use, each method has its pros and cons (see comparison)



## Local Surface Analysis

- Cloud of point samples describes underlying (manifold) surface
- We need:
  - Mechanisms for locally approximating the surface ⇨ MLS approach
  - Fast estimation of tangent plane and curvature ⇨ principal component analysis of local neighborhood

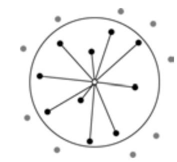
# Neighborhood

- No explicit connectivity between samples (as with triangle meshes)
- Replace geodesic proximity with spatial proximity (requires sufficiently high sampling density!)
- Compute neighborhood according to Euclidean distance



# Neighborhood

- K-nearest neighbors

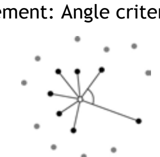


  - Can be quickly computed using spatial data-structures (e.g. kd-tree, octree, bsp-tree)
  - Requires isotropic point distribution



# Neighborhood

- Improvement: Angle criterion (Linsen)



  - Project points onto tangent plane
  - Sort neighbors according to angle
  - Include more points if angle between subsequent points is above some threshold



# Neighborhood

- Local Delaunay triangulation (Floater)

  - Project points into tangent plane
  - Compute local Voronoi diagram



# Covariance Analysis

- Covariance matrix of local neighborhood N:

$$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{i_1} - \bar{\mathbf{p}} \\ \Lambda \\ \mathbf{p}_{i_n} - \bar{\mathbf{p}} \end{bmatrix}^T \cdot \begin{bmatrix} \mathbf{p}_{i_1} - \bar{\mathbf{p}} \\ \Lambda \\ \mathbf{p}_{i_n} - \bar{\mathbf{p}} \end{bmatrix}, \qquad i_j \in N$$

- with centroid $\bar{\mathbf{p}} = \frac{1}{|N|} \sum_{i \in N} \mathbf{p}_i$



# Covariance Analysis

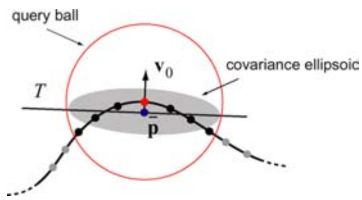
- Consider the eigenproblem:

$$\mathbf{C} \cdot \mathbf{v}_l = \lambda_l \cdot \mathbf{v}_l, \qquad l \in \{0,1,2\}$$

- C is a 3x3, positive semi-definite matrix
  - ⇨ All eigenvalues are real-valued
  - ⇨ The eigenvector with smallest eigenvalue defines the least-squares plane through the points in the neighborhood, i.e. approximates the surface normal

## Covariance Analysis

- Covariance ellipsoid spanned by the eigenvectors scaled with corresponding eigenvalue






## Covariance Analysis

- The total variation is given as:

$$\sum_{i \in N} \left| \mathbf{p}_i - \overline{\mathbf{p}} \right|^2 = \lambda_0 + \lambda_1 + \lambda_2$$

- We define surface variation as:

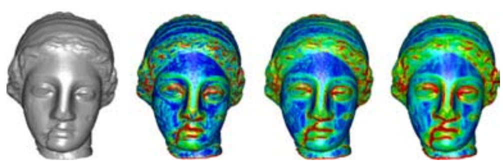
$$\sigma_n(\mathbf{p}) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}, \qquad \lambda_0 \le \lambda_1 \le \lambda_2$$

  - Measures the fraction of variation along the surface normal, i.e. quantifies how strong the surface deviates from the tangent plane ⇨ estimate for curvature



## Covariance Analysis

- Comparison with curvature:



original mean curvature variation n=20 variation n=50



## Surface Simplification

- Hierarchical clustering
- Iterative simplification
- Particle simulation



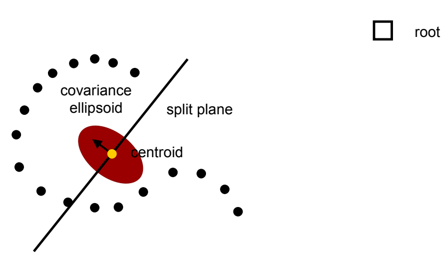
## Hierarchical Clustering

- Top-down approach using binary space partition:
- Split the point cloud if:
  - Size is larger than user-specified maximum or
  - Surface variation is above maximum threshold
- Split plane defined by centroid and axis of greatest variation (= eigenvector of covariance matrix with largest associated eigenvector)
- Leaf nodes of the tree correspond to clusters
- Replace clusters by centroid



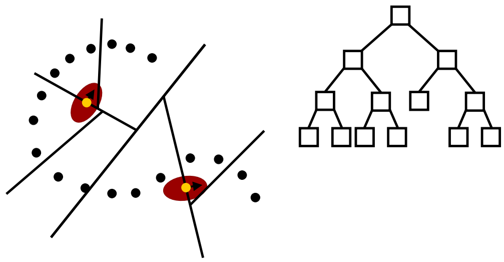
## Hierarchical Clustering

- 2D example



## Hierarchical Clustering

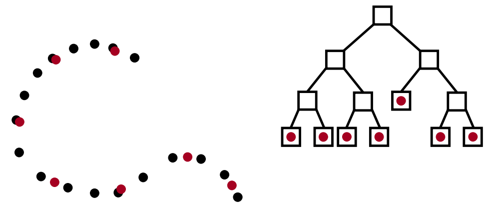
- 2D example





## Hierarchical Clustering

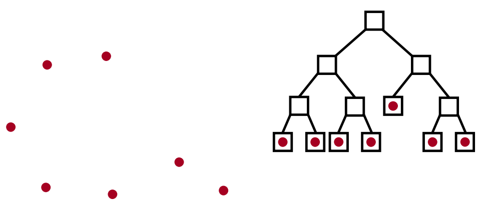
- 2D example





## Hierarchical Clustering

- 2D example





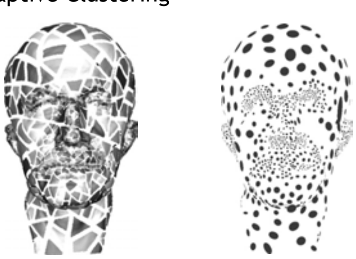
## Hierarchical Clustering

43 Clusters 436 Clusters 4,280 Clusters



## Hierarchical Clustering

- Adaptive Clustering



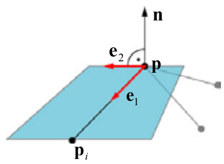


## Iterative Simplification

- Iteratively contracts point pairs
  - ⇨ Each contraction reduces the number of points by one
- Contractions are arranged in priority queue according to quadric error metric (Garland and Heckbert)
- Quadric measures cost of contraction and determines optimal position for contracted sample
- Equivalent to QSlim except for definition of approximating planes

## Iterative Simplification

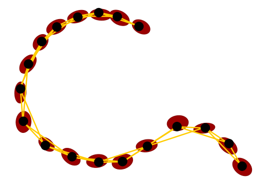
- Quadric measures the squared distance to a set of planes defined over *edges* of neighborhood
  - plane spanned by vectors $\mathbf{e}_1 = \mathbf{p}_i - \mathbf{p}$ and $\mathbf{e}_2 = \mathbf{e}_1 \times \mathbf{n}$






## Iterative Simplification

- 2D example



- Compute initial point-pair contraction candidates
- Compute fundamental quadrics
- Compute edge costs



## Iterative Simplification

- 2D example

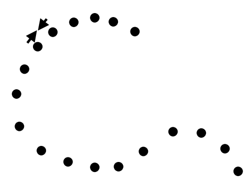
priority queue

| edge | cost |
|---|---|
| 6 | 0.02 |
| 2 | 0.03 |
| 14 | 0.04 |
| 5 | 0.04 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.22 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.44 |
| 4 | 0.56 |



## Iterative Simplification

- 2D example



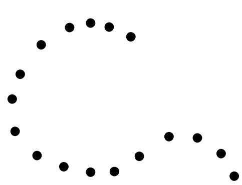
priority queue

| edge | cost |
|---|---|
| 6 | 0.02 |
| 2 | 0.03 |
| 14 | 0.04 |
| 5 | 0.04 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.22 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.44 |
| 4 | 0.56 |



## Iterative Simplification

- 2D example



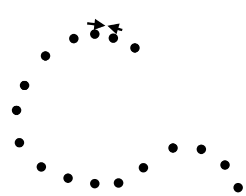
priority queue

| edge | cost |
|---|---|
| 6 | 0.02 |
| 2 | 0.03 |
| 14 | 0.04 |
| 5 | 0.06 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.23 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.49 |
| 4 | 0.56 |



## Iterative Simplification

- 2D example

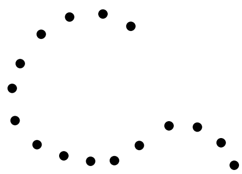


priority queue

| edge | cost |
|---|---|
| 2 | 0.03 |
| 14 | 0.04 |
| 5 | 0.06 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.23 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.49 |
| 4 | 0.56 |

## Iterative Simplification

- 2D example



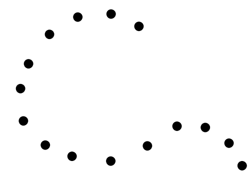
priority queue

| edge | cost |
|---|---|
| 2 | 0.03 |
| 14 | 0.04 |
| 5 | 0.06 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.23 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.49 |
| 4 | 0.56 |



## Iterative Simplification

- 2D example



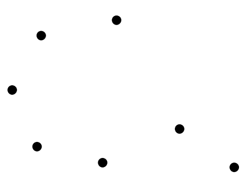
priority queue

| edge | cost |
|---|---|
| 14 | 0.04 |
| 5 | 0.06 |
| 9 | 0.09 |
| 1 | 0.11 |
| 13 | 0.13 |
| 3 | 0.23 |
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.49 |
| 4 | 0.56 |



## Iterative Simplification

- 2D example



priority queue

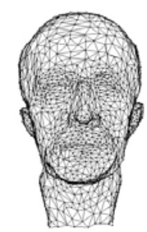
| edge | cost |
|---|---|
| 11 | 0.27 |
| 10 | 0.36 |
| 7 | 0.49 |
| 4 | 0.56 |



## Iterative Simplification

original model (296,850 points)

simplified model (2,000 points)



remaining point pair contraction candidates



## Particle Simulation

- Resample surface by distributing particles on the surface
- Particles move on surface according to inter-particle repelling forces
- Particle relaxation terminates when equilibrium is reached (requires damping)
- Can also be used for up-sampling!



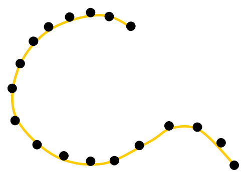
## Particle Simulation

- Initialization
  - Randomly spread particles
- Repulsion
  - Linear repulsion force $F_i(\mathbf{p}) = k(r - \|\mathbf{p}-\mathbf{p}_i\|)\cdot(\mathbf{p}-\mathbf{p}_i)$

  ⇨ only need to consider neighborhood of radius r
- Projection
  - Keep particles on surface by projecting onto tangent plane of closest point
  - Apply full MLS projection at end of simulation

## Particle Simulation

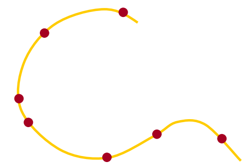
- 2D example





## Particle Simulation

- 2D example
  - Initialization
    - randomly spread particles

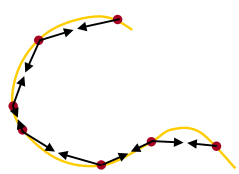




## Particle Simulation

- 2D example
  - Initialization
    - randomly spread particles
  - Repulsion
    - linear repulsion force

$$F_i(\mathbf{p}) = k(r - \|\mathbf{p}-\mathbf{p}_i\|)\cdot(\mathbf{p}-\mathbf{p}_i)$$

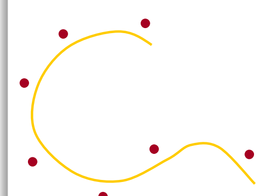




## Particle Simulation

- 2D example
  - Initialization
    - randomly spread particles
  - Repulsion
    - linear repulsion force

$$F_i(\mathbf{p}) = k(r - \|\mathbf{p}-\mathbf{p}_i\|)\cdot(\mathbf{p}-\mathbf{p}_i)$$



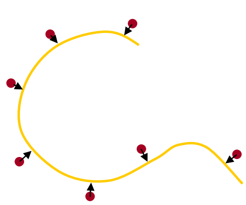


## Particle Simulation

- 2D example
  - Initialization
    - randomly spread particles
  - Repulsion
    - linear repulsion force

$$F_i(\mathbf{p}) = k(r - \|\mathbf{p}-\mathbf{p}_i\|)\cdot(\mathbf{p}-\mathbf{p}_i)$$

  - Projection
    - project particles onto surface



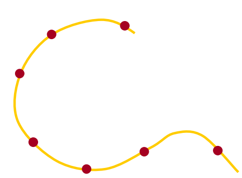


## Particle Simulation

- 2D example
  - Initialization
    - randomly spread particles
  - Repulsion
    - linear repulsion force

$$F_i(\mathbf{p}) = k(r - \|\mathbf{p}-\mathbf{p}_i\|)\cdot(\mathbf{p}-\mathbf{p}_i)$$

  - Projection
    - project particles onto surface



## Particle Simulation

- Adaptive simulation
  - Adjust repulsion radius according to surface variation
    ⇨ more samples in regions of high variation

variation estimation

simplified model (3,000 points)



## Particle Simulation

- User-controlled simulation
  - Adjust repulsion radius according to user input

uniform

original

selective



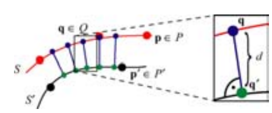
## Measuring Error

- Measure the distance between two point-sampled surfaces using a sampling approach
- Maximum error: $\Delta_{\max}(S, S') = \max_{\mathbf{q} \in Q} d(\mathbf{q}, S')$
  ⇨ Two-sided Hausdorff distance
- Mean error: $\Delta_{\text{avg}}(S, S') = \frac{1}{|Q|} \sum_{\mathbf{q} \in Q} d(\mathbf{q}, S')$
  ⇨ Area-weighted integral of point-to-surface distances
- $Q$ is an up-sampled version of the point cloud that describes the surface $S$



## Measuring Error

- $d(\mathbf{q}, S')$ measures the distance of point $\mathbf{q}$ to surface $S'$ using the MLS projection operator with linear basis functions






## Measuring Error

original

simplified

upsampled

error



## Comparison

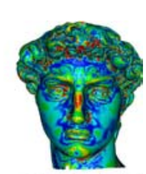
- Error estimate for Michelangelo's David simplified from 2,000,000 points to 5,000 points

$\Delta_{avg} = 6.14 \cdot 10^{-4}$ $\Delta_{max} = 0.0046$

adaptive hierarchical clustering

$\Delta_{avg} = 5.43 \cdot 10^{-4}$ $\Delta_{max} = 0.0052$

iterative simplification

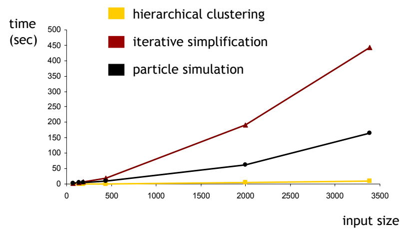


$\Delta_{avg} = 5.69 \cdot 10^{-4}$ $\Delta_{max} = 0.0061$

particle simulation

## Comparison

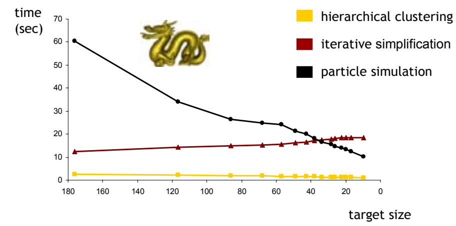
- Execution time as a function of input model size (reduction to 1%)






## Comparison

- Execution time as a function of target model size (input: dragon, 535,545 points)





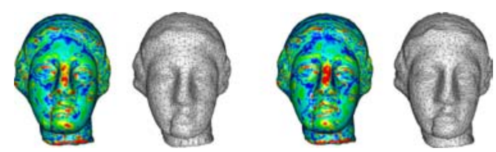

## Comparison

- Summary

| | Efficiency | Surface Error | Control | Implementation |
|---|---|---|---|---|
| Hierarchical Clustering | + | - | - | + |
| Iterative Simplification | - | + | o | o |
| Particle Simulation | o | + | + | - |



## Point-based vs. Mesh Simplification



point-based simplification with subsequent mesh reconstruction

mesh reconstruction with subsequent mesh simplification (QSlim)

⇨ point-based simplification saves an expensive surface reconstruction on the dense point cloud!



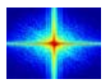
## References

- Pauly, Gross: *Efficient Simplification of Point-sampled Surfaces*, IEEE Visualization 2002
- Shaffer, Garland: *Efficient Adaptive Simplification of Massive Meshes*, IEEE Visualization 2001
- Garland, Heckbert: *Surface Simplification using Quadric Error Metrics*, SIGGRAPH 1997
- Turk: *Re-Tiling Polygonal Surfaces*, SIGGRAPH 1992
- Alexa et al. *Point Set Surfaces*, IEEE Visualization 2001

# Eurographics 2003

## Spectral Processing of Point-Sampled Geometry

Markus Gross



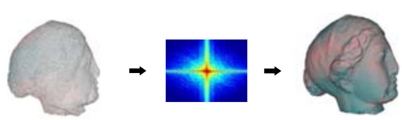
## Overview

- Introduction
- Fourier transform
- Spectral processing pipeline
- Applications
  - Spectral filtering
  - Adaptive subsampling
- Summary



## Introduction

- Idea: Extend the Fourier transform to manifold geometry



⇨ Spectral representation of point-based objects

⇨ Powerful methods for digital geometry processing



## Introduction

- Applications:
  - Spectral filtering:
    - Noise removal
    - Microstructure analysis
    - Enhancement
  - Adaptive resampling:
    - Complexity reduction
    - Continuous LOD



## Fourier Transform

- 1D example:

$$\underbrace{X_n}_{\text{output signal}} = \sum_{k=1}^{N} \underbrace{x_k}_{\text{input signal}} \underbrace{e^{-j2\pi\frac{nk}{N}}}_{\text{spectral basis function}}$$

- Benefits:
  - Sound concept of frequency
  - Extensive theory
  - Fast algorithms



## Fourier Transform

- Requirements:
  - Fourier transform defined on Euclidean domain
    - ⇨we need a global parameterization
  - Basis functions are eigenfunctions of Laplacian operator
    - ⇨requires regular sampling pattern so that basis functions can be expressed in analytical form (fast evaluation)
- Limitations:
  - Basis functions are globally defined
    - ⇨Lack of local control

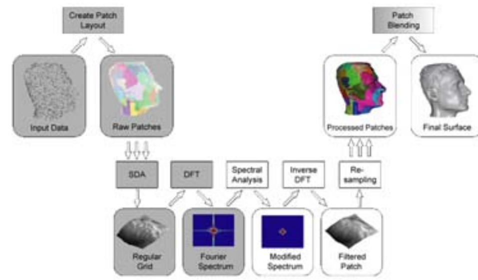
# Approach

- Split model into patches that:
  - are parameterized over the unit-square
    - ⇨ mapping must be continuous and should minimize distortion
  - are re-sampled onto a regular grid
    - ⇨ adjust sampling rate to minimize information loss
  - provide sufficient granularity for intended application (local analysis)

⇨ process each patch individually and blend processed patches



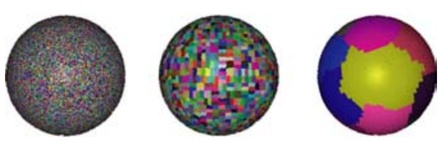
# Spectral Pipeline






# Patch Layout Creation

Clustering ⇨ Optimization



Samples ⇨ Clusters ⇨ Patches



# Patch Layout Creation

- Iterative, local optimization method
- Merge patches according to quality metric:

$$\Phi = \Phi_S \cdot \Phi_{NC} \cdot \Phi_B \cdot \Phi_{Reg}$$

$\Phi_S$ ⇨ patch Size

$\Phi_{NC}$ ⇨ curvature

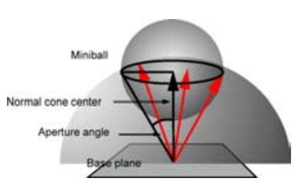
$\Phi_B$ ⇨ patch boundary

$\Phi_{Reg}$ ⇨ spring energy regularization



# Patch Layout Creation

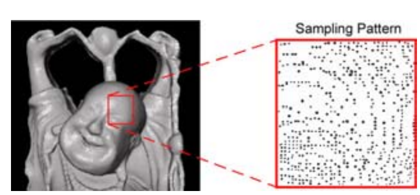
- Parameterize patches by orthogonal projection onto base plane
- Bound normal cone to control distortion of mapping using smallest enclosing sphere





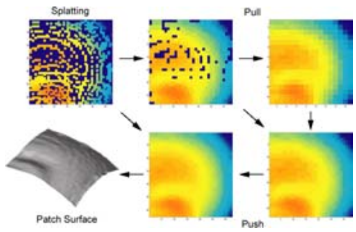

# Patch Resampling

- Patches are irregularly sampled:



## Patch Resampling

- Resample patch onto regular grid using hierarchical push-pull filter (scattered data approximation)






## Spectral Analysis

- 2D discrete Fourier transform (DFT)
  - ⇨ Direct manipulation of spectral coefficients
- Filtering as convolution:

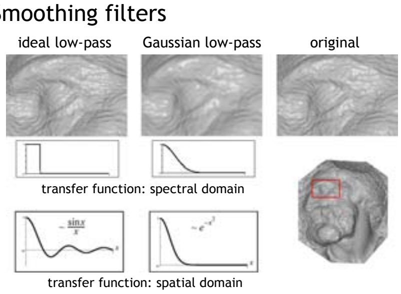
$$F(x \otimes y) = F(x) \cdot F(y)$$

  - ⇨ Convolution: $O(N^2)$ ⇨ multiplication: $O(N)$
- Inverse Fourier transform
  - ⇨ Filtered patch surface



## Spectral Filters

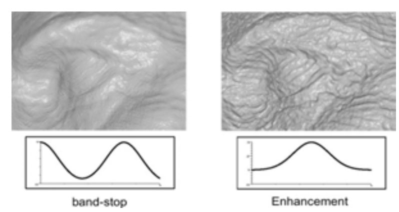
- Smoothing filters






## Spectral Filters

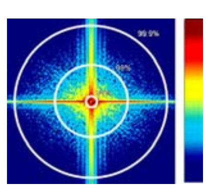
- Microstructure analysis and enhancement






## Spectral Resampling

- Low-pass filtering
  - ⇨ Band-limitation
- Regular Resampling
  - ⇨ Optimal sampling rate (sampling theorem)
  - ⇨ Error control (Parseval's theorem)

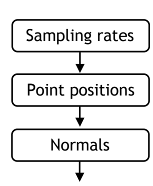


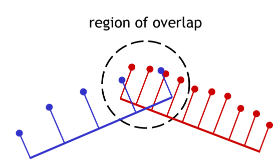
Power Spectrum



## Reconstruction

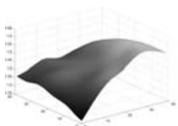
- Filtering can lead to discontinuities at patch boundaries
  - ⇨ Create patch overlap, blend adjacent patches



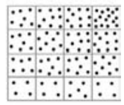
# Reconstruction

- Blending the sampling rate



blended sampling rate in region of patch overlap →

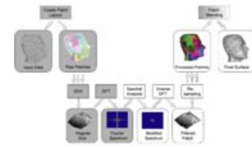
discretized sampling rate on regular grid →

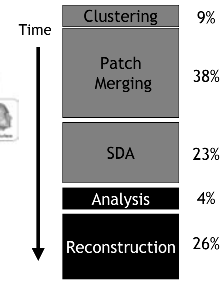


pre-computed sampling patterns



# Timings

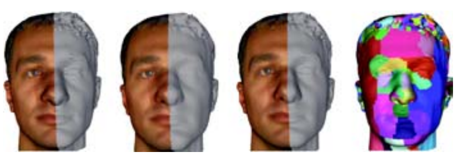







# Applications

- Surface Restoration

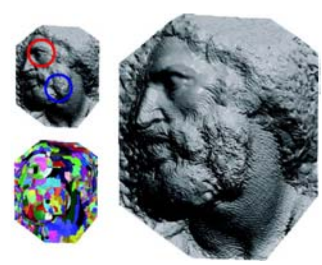


Original Gaussian low-pass Wiener filter Patch layout



# Applications

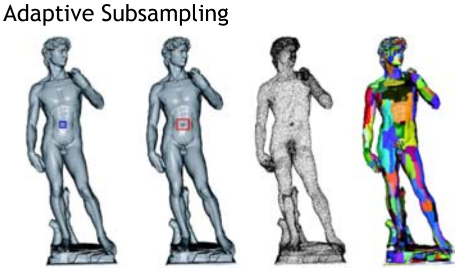
- Interactive filtering





# Applications

- Adaptive Subsampling



4,128,614 pts. = 100% 287,163 pts. = 6.9%



# Summary

- Versatile spectral decomposition of point-based models
- Effective filtering
- Adaptive resampling
- Efficient processing of large point-sampled models

# Reference

- Pauly, Gross: *Spectral Processing of Point-sampled Geometry*, SIGGRAPH 2001

## Eurographics 2003

pointshop

An Interactive System for Point-based Surface Editing

## Overview

- Introduction
- Pointshop3D System Components
  - Point Cloud Parameterization
  - Resampling Scheme
  - Editing Operators
- Summary

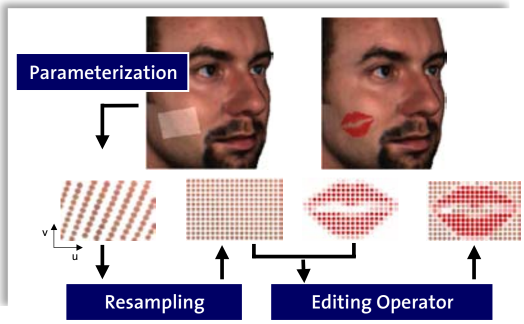


## PointShop3D

- Interactive system for point-based surface editing
- Generalizes 2D photo editing concepts and functionality to 3D point-sampled surfaces
- Uses 3D surface pixels (*surfels*) as versatile display and modeling primitive



## Concept






## Key Components

- Point cloud parameterization $\Phi$
  - brings surface and brush into common reference frame
- Dynamic resampling $\Psi$
  - creates one-to-one correspondence of surface and brush samples
- Editing operator $\Omega$
  - combines surface and brush samples

$$\underbrace{S'}_{\text{modified surface}} = \Omega(\Psi(\Phi(\underbrace{S}_{\text{original surface}})), \Psi(\underbrace{B}_{\text{brush}}))$$



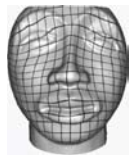
## Parameterization

- Constrained minimum distortion parameterization of point clouds

$$\mathbf{u} \in [0,1]^2 \Rightarrow X(\mathbf{u}) = \begin{bmatrix} x(\mathbf{u}) \\ y(\mathbf{u}) \\ z(\mathbf{u}) \end{bmatrix} = \mathbf{x} \in P \subset R^3$$

## Parameterization

contraints = matching
of feature points



minimum distortion =
maximum smoothness



## Parameterization

- Find mapping X that minimizes objective function:

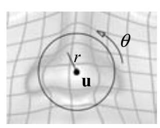
$$C(X) = \underbrace{\sum_{j \in M} (X(\overset{\text{brush points}}{\mathbf{p}_j}) - \overset{\text{surface points}}{\mathbf{x}_j})^2}_{\text{fitting constraints}} + \varepsilon \underbrace{\int_P \gamma(\mathbf{u})\, d\mathbf{u}}_{\text{distortion}}$$



## Parameterization

- Measuring distortion

$$\gamma(\mathbf{u}) = \int_\theta \left( \frac{\partial^2}{\partial r^2} X_{\mathbf{u}}(\theta, r) \right)^2 d\theta$$




- Integrates squared curvature using local polar re-parameterization

$$X_{\mathbf{u}}(\theta, r) = X\left( \mathbf{u} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \right)$$



## Parameterization

- Discrete formulation:

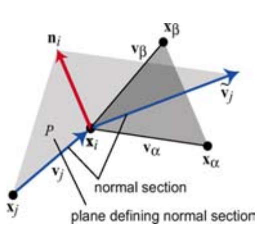
$$\tilde{C}(U) = \sum_{j \in M} (\mathbf{p}_j - \mathbf{u}_j)^2 + \varepsilon \sum_{i=1}^{n} \sum_{j \in N_i} \left( \frac{\partial U(\mathbf{x}_i)}{\partial \mathbf{v}_j} - \frac{\partial U(\mathbf{x}_i)}{\partial \tilde{\mathbf{v}}_j} \right)^2$$

- Approximation: mapping is piecewise linear



## Parameterization

- Directional derivatives as extension of divided differences based on k-nearest neighbors

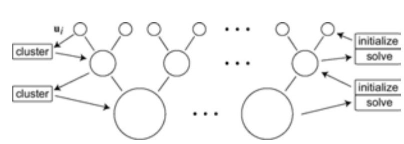





## Parameterization

- Multigrid solver for efficient computation of resulting sparse linear least squares problem

$$\tilde{C}(U) = \sum_j \left( \mathbf{b}_j - \sum_{i=1}^{n} a_{j,i} \mathbf{u}_i \right)^2 = \|\mathbf{b} - A\mathbf{u}\|^2$$



# Reconstruction

- Parameterized scattered data approximation

fitting functions

weight functions

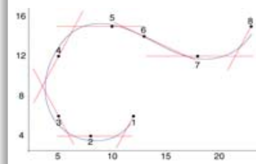
$$X(\mathbf{u}) = \frac{\sum_i \Phi_i(\mathbf{u}) r_i(\mathbf{u})}{\sum_i r_i(\mathbf{u})}$$

normalization factor

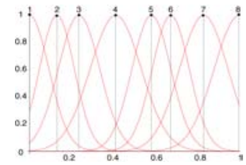
- Fitting functions
  - Compute local fitting functions using local parameterizations
  - Map to global parameterization using global parameter coordinates of neighboring points



# Reconstruction




reconstruction with linear fitting functions




weight functions in parameter space



# Reconstruction

- Reconstruction with linear fitting functions is equivalent to surface splatting!
  - ⇨ we can use the surface splatting renderer to reconstruct our surface function (see chapter on rendering)
  - This provides:
    - Fast evaluation
    - Anti-aliasing (Band-limit the weight functions before sampling using Gaussian low-pass filter)
  - Distortions of splats due to parameterization can be computed efficiently using local affine mappings



# Sampling

- Three sampling strategies:
  - Resample the brush, i.e., sample at the original surface points
  - Resample the surface, i.e., sample at the brush points
  - Adaptive resampling, i.e., sample at surface or brush points depending on the respective sampling density



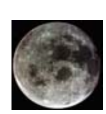
# Editing Operators

- Painting
  - Texture, material properties, transparency

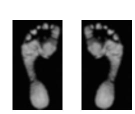


# Editing Operators

- Sculpting
  - Carving, normal displacement



texture map

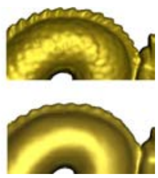


displacement maps

carved and texture mapped point-sampled surface

# Editing Operators

- Filtering
  - Scalar attributes, geometry





# Summary

- Pointshop3D provides sophisticated editing operations on point-sampled surfaces
  ⇨ points are a versatile and powerful modeling primitive
- Limitation: only works on “clean” models
  - sufficiently high sampling density
  - no outliers
  - little noise
  ⇨ requires model cleaning (integrated or as pre-process)



# Reference

- Zwicker, Pauly, Knoll, Gross: *Pointshop3D: An interactive system for Point-based Surface Editing*, SIGGRAPH 2002

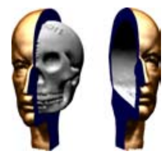
pointshop

- check out:

**www.pointshop3D.com**

# Eurographics 2003

## Shape Modeling

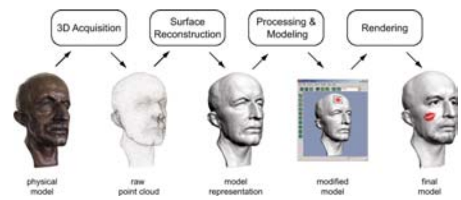


Mark Pauly

---

## Motivation

- 3D content creation pipeline






---

## Motivation

- Surface representations
  - Implicit surfaces
    - Level sets
    - Radial basis functions → + Extreme deformations, + Changes of topology
    - Algebraic surfaces
  - Parametric surfaces
    - Polygonal meshes
    - Subdivision surfaces → + Sharp features, + Efficient rendering, + Intuitive Editing
    - NURBS



---

## Motivation

- Surface representations
  - Implicit surfaces
    - Level sets
    - Radial basis functions
    - Algebraic surfaces
  - Parametric surfaces
    - Polygonal meshes
    - Subdivision surfaces
    - Nurbs

Hybrid Representation

- Explicit cloud of point samples
- Implicit dynamic surface model



---

## Motivation

- Point cloud representation
  - Minimal consistency requirements for extreme deformations (dynamic re-sampling)
  - Fast inside/outside classification for boolean operations and collision detection
  - Explicit modeling and rendering of sharp feature curves
  - Integrated, intuitive editing of shape and appearance

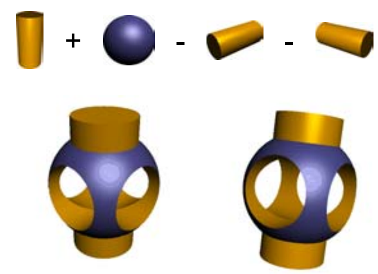


---

## Interactive Modeling

- Interactive design and editing of point-sampled models
  - Shape Modeling
    - Boolean operations
    - Free-form deformation
  - Appearance Modeling
    - Painting & texturing
    - Embossing & engraving

## Boolean Operations






## Boolean Operations

- Create new shapes by combining existing models using union, intersection, or difference operations
- Powerful and flexible editing paradigm mostly used in industrial design applications (CAD/CAM)



## Boolean Operations

- Easily performed on implicit representations
  - Requires simple computations on the distance function
- Difficult for parametric surfaces
  - Requires surface-surface intersection
- Topological complexity of resulting surface depends on geometric complexity of input models



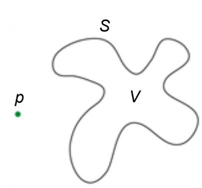
## Boolean Operations

- Point-Sampled Geometry
  - Classification
    - Inside-outside test using signed distance function induced by MLS projection
  - Sampling
    - Compute exact intersection of two MLS surfaces to sample the intersection curve
  - Rendering
    - Accurate depiction of sharp corners and creases using point-based rendering



## Boolean Operations

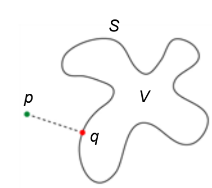
- Classification:
  - given a smooth, closed surface $S$ and point $p$. Is $p$ inside or outside of the volume $V$ bounded by $S$?





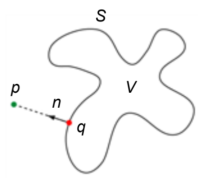

## Boolean Operations

- Classification:
  - given a smooth, closed surface $S$ and point $p$. Is $p$ inside or outside of the volume $V$ bounded by $S$?
  1. find closest point $q$ on $S$



## Boolean Operations

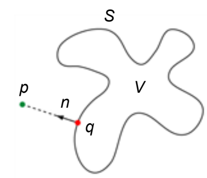
- Classification:
  - given a smooth, closed surface $S$ and point $p$. Is $p$ inside or outside of the volume $V$ bounded by $S$?
  1. find closest point $q$ on $S$
  2. $d=(p-q)\cdot n$ defines signed distance of $p$ to $S$






## Boolean Operations

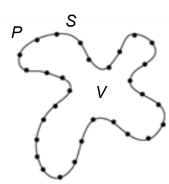
- Classification:
  - given a smooth, closed surface $S$ and point $p$. Is $p$ inside or outside of the volume $V$ bounded by $S$?
  1. find closest point $q$ on $S$
  2. $d=(p-q)\cdot n$ defines signed distance of $p$ to $S$
  3. classify $p$ as
     - inside $V$, if $d < 0$
     - outside $V$, if $d > 0$






## Boolean Operations

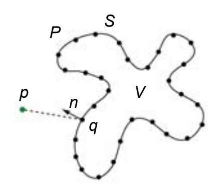
- Classification:
  - represent smooth surface $S$ by point cloud $P$






## Boolean Operations

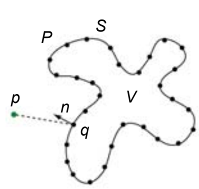
- Classification:
  - represent smooth surface $S$ by point cloud $P$
  1. find closest point $q$ in $P$






## Boolean Operations

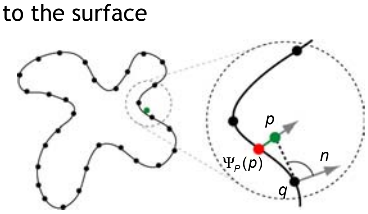
- Classification:
  - represent smooth surface $S$ by point cloud $P$
  1. find closest point $q$ in $P$
  2. classify $p$ as
     - inside $V$, if $(p-q)\cdot n < 0$
     - outside $V$, if $(p-q)\cdot n > 0$





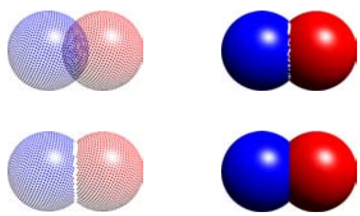

## Boolean Operations

- Classification:
  - apply full MLS projection for points close to the surface



# Boolean Operations

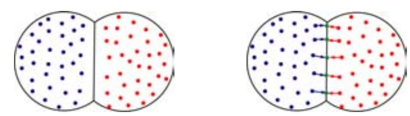
- Sampling the intersection curve





# Boolean Operations

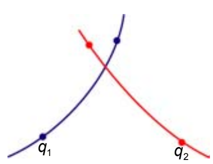
- Newton scheme:
  1. identify pairs of closest points





# Boolean Operations

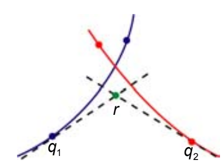
- Newton scheme:
  1. identify pairs of closest points






# Boolean Operations

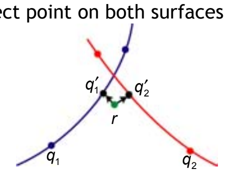
- Newton scheme:
  1. identify pairs of closest points
  2. compute closest point on intersection of tangent spaces






# Boolean Operations

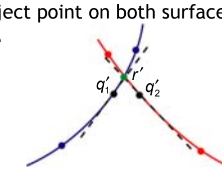
- Newton scheme:
  1. identify pairs of closest points
  2. compute closest point on intersection of tangent spaces
  3. re-project point on both surfaces





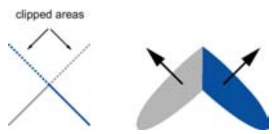

# Boolean Operations

- Newton scheme:
  1. identify pairs of closest points
  2. compute closest point on intersection of tangent spaces
  3. re-project point on both surfaces
  4. iterate



## Boolean Operations

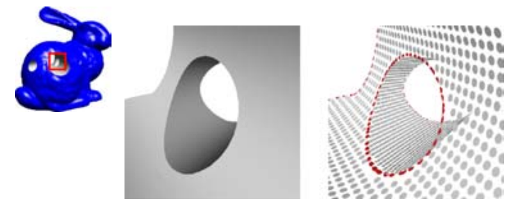
- Rendering sharp creases
  - represent points on intersection curve with two surfels that mutually clip each other






## Boolean Operations

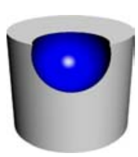
- Rendering sharp creases





## Boolean Operations

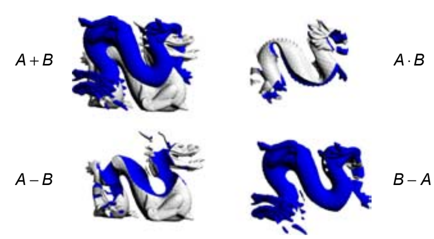
- Rendering sharp creases
  - easily extended to handle corners by allowing multiple clipping





## Boolean Operations

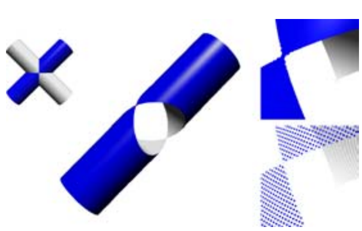
- Boolean operations can create intricate shapes with complex topology






## Boolean Operations

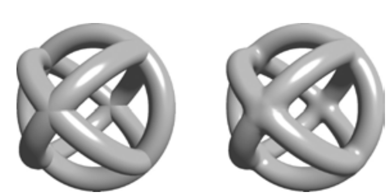
- Singularities lead to numerical instabilities (intersection of almost parallel planes)





## Boolean Operations

- Sharp creases can be blended using oriented particles (Szeliski, Tonnesen)

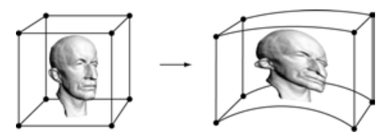


## Free-form Deformation



## Free-form Deformation

- Smooth deformation field $F: \mathbf{R}^3 \rightarrow \mathbf{R}^3$ that warps 3D space
- Can be applied directly to point samples





## Free-form Deformation

- How to define the deformation field?
  - ⇨ Painting metaphor
- How to detect and handle self-intersections?
  - ⇨ Point-based collision detection, boolean union, particle-based blending
- How the handle strong distortions?
  - ⇨ Dynamic re-sampling

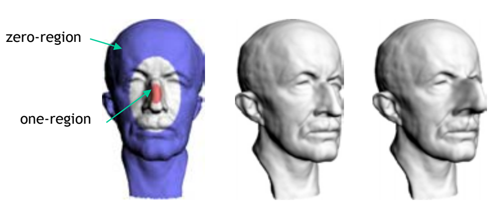


## Free-form Deformation

- Intuitive editing paradigm using painting metaphor
  - Define rigid surface part (zero-region) and handle (one-region) using interactive painting tool
  - Displace handle using combination of translation and rotation
  - Create smooth blend towards zero-region



## Free-form Deformation



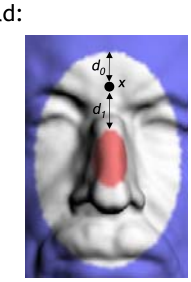

original surface

deformed surface



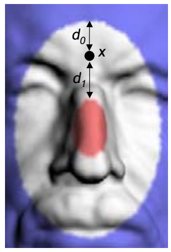
## Free-form Deformation

- Definition of deformation field:
  - Continuous scale parameter $t_x$
    - $t_x = \beta(d_0 / (d_0 + d_1))$
    - $d_0$ : distance of $x$ to zero-region
    - $d_1$ : distance of $x$ to one-region
  - Blending function
    - $\beta : [0,1] \rightarrow [0,1]$
    - $\beta \in C^0$, $\beta(0) = 0$, $\beta(1) = 1$
  - $t_x = 0$ if $x$ in zero-region
  - $t_x = 1$ if $x$ in one-region



# Free-form Deformation

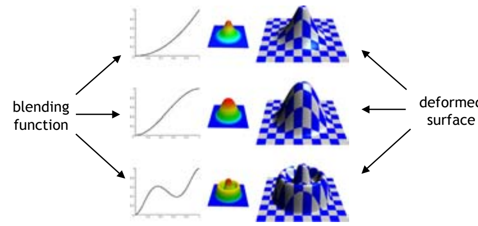
- Definition of deformation field:
  - Deformation function
    - $F(x) = F_T(x) + F_R(x)$
  - Translation
    - $F_T(x) = x + t_x \cdot v$
  - Rotation
    - $F_R(x) = M(t_x) \cdot x$






# Free-form Deformation

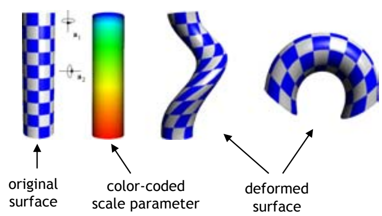
- Translation for three different blending functions






# Free-form Deformation

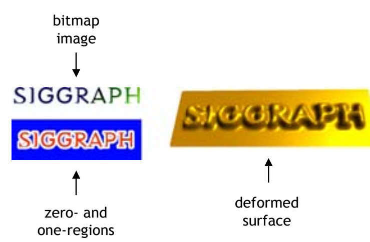
- Rotational deformation along two different rotation axes






# Free-form Deformation

- Embossing effect





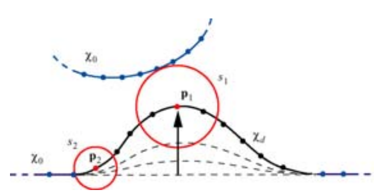

# Collision Detection

- Deformations can lead to self-intersections
- Apply boolean inside/outside classification to detect collisions
- Restricted to collisions between deformable region and zero-region to ensure efficient computations



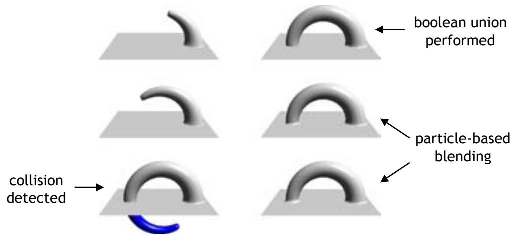
# Collision Detection

- Exploiting temporal coherence



## Collision Detection

- Interactive modeling session





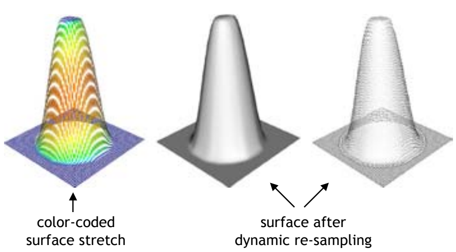

## Dynamic Sampling

- Large model deformations can lead to strong surface distortions
- Requires adaptation of the sampling density
- Dynamic insertion and deletion of point samples



## Dynamic Sampling

- Surface distortion varies locally





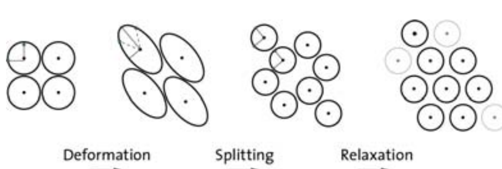

## Dynamic Sampling

1. Measure local surface stretch from first fundamental form
2. Split samples that exceed stretch threshold
3. Regularize distribution by relaxation
4. Interpolate scalar attributes



## Dynamic Sampling

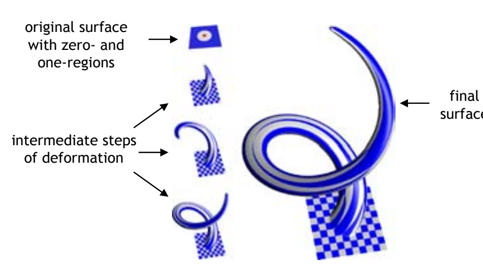
- 2D illustration






## Free-form Deformation

- Interactive modeling session with dynamic sampling



# Results

- 3D shape modeling functionality has been integrated into Pointshop3D to create a complete system for point-based shape and appearance modeling
  - Boolean operations
  - Free-form deformation
  - Painting & texturing
  - Sculpting
  - Filtering
  - Etc.



# Results

- Ab-initio design of an Octopus
  - Free-form deformation with dynamic sampling from 69,706 to 295,222 points



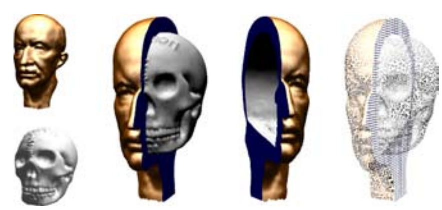
# Results

- Modeling with synthetic and scanned data
  - Combination of free-form deformation with collision detection, boolean operations, particle-based blending, embossing and texturing



# Results

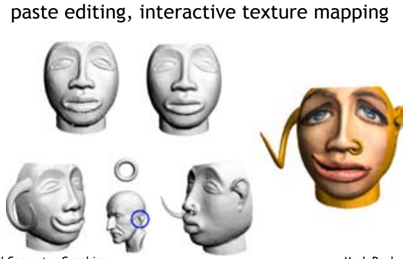
- Boolean operations on scanned data
  - Irregular sampling pattern, low resolution models





# Results

- Interactive modeling with scanned data
  - noise removal, free-form deformation, cut-and-paste editing, interactive texture mapping





# Conclusion

- Points are a versatile shape modeling primitive
  - Combines advantages of implicit and parametric surfaces
  - Integrates boolean operations and free-form deformation
  - Dynamic restructuring
  - Time and space efficient implementations

## Conclusion

- Complete and versatile point-based 3D shape and appearance modeling system
  - Directly applicable to scanned data
  - Suitable for low-cost 3D content creation and rapid proto-typing



## References


- Pauly: Point Primitives for Interactive Modeling and Processing of 3D Geometry, PhD Thesis, ETH Zurich, 2003
- Pauly, Keiser, Kobbelt, Gross: Shape Modeling with Point-sampled Geometry, SIGGRAPH 03
- Pauly, Kobbelt, Gross: Multiresolution Modeling with Point-sampled Geometry, ETH Technical Report, 2002
- Zwicker, Pauly, Knoll, Gross: Pointshop3D: An Interactive System for Point-based Surface Editing, SIGGRAPH 02
- Adams, Dutre: Boolean Operations on Surfel-Bounded Solids, SIGGRAPH 03
- Szeliski, Tonnesen: Surface Modeling with Oriented Particle Systems, SIGGRAPH 92
- www.pointshop3d.com