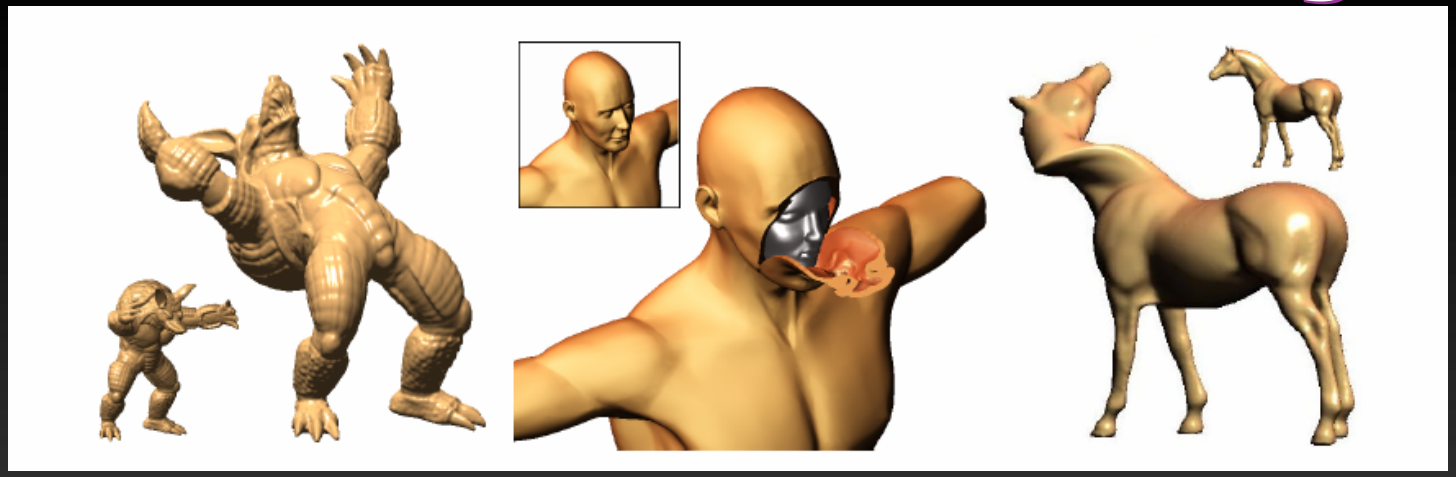
Programmable Shaders for Deformation Rendering



Carlos D. Correa, Deborah Silver

Rutgers, The State University of New Jersey

Motivation

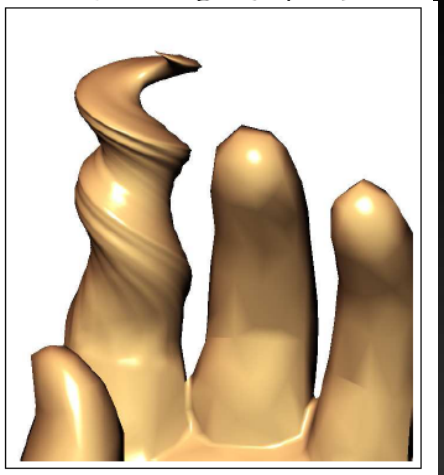
- We present a different way of obtaining mesh deformation.
- Not a modeling, but a rendering problem.
- We aim towards generation of high-quality deformations on meshes of arbitrary tessellation.
- Can we define a “shader” for doing deformation as a rendering step? Advantages:
  - Independent of mesh resolution
  - Allows cuts without the need for re-meshing
  - Incorporates other rendering effects, such as shadows, translucency,…

Effects of Mesh Resolution

2,874 triangles (Explicit)

18,905 triangles (Explicit)

52,463 triangles (Explicit)

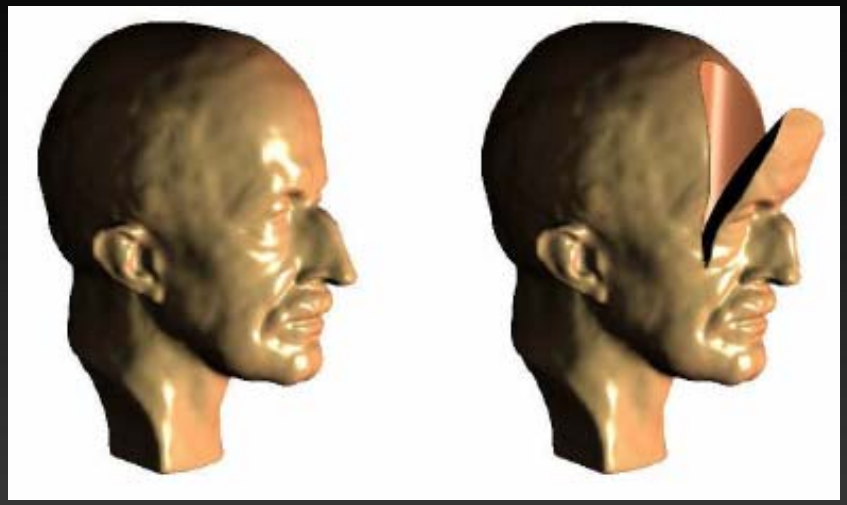


2,874 triangles (Our approach)

18,905 (Our approach)

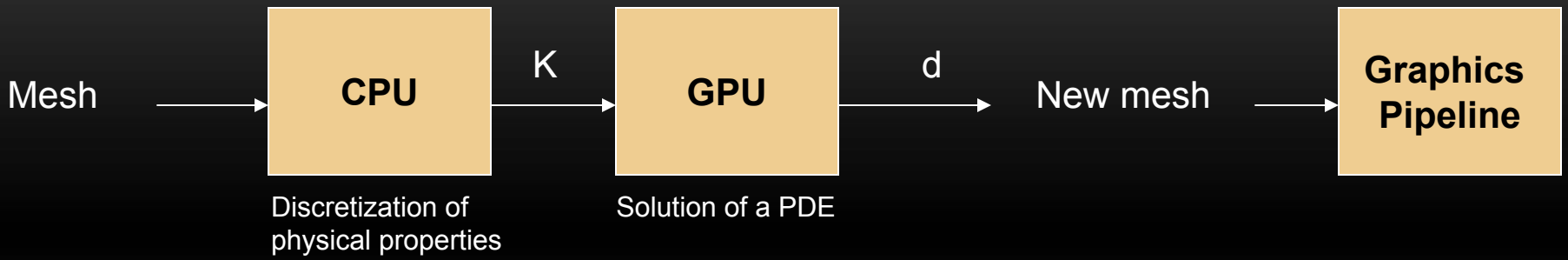
52,463 triangles (Our approach)

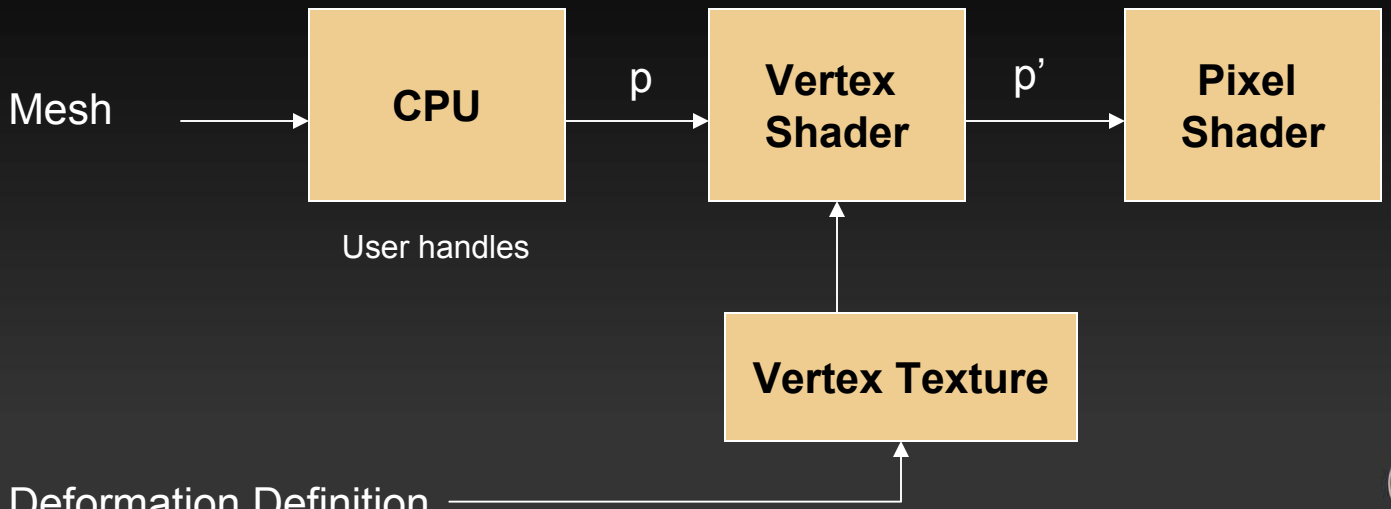
Simulation of Cuts



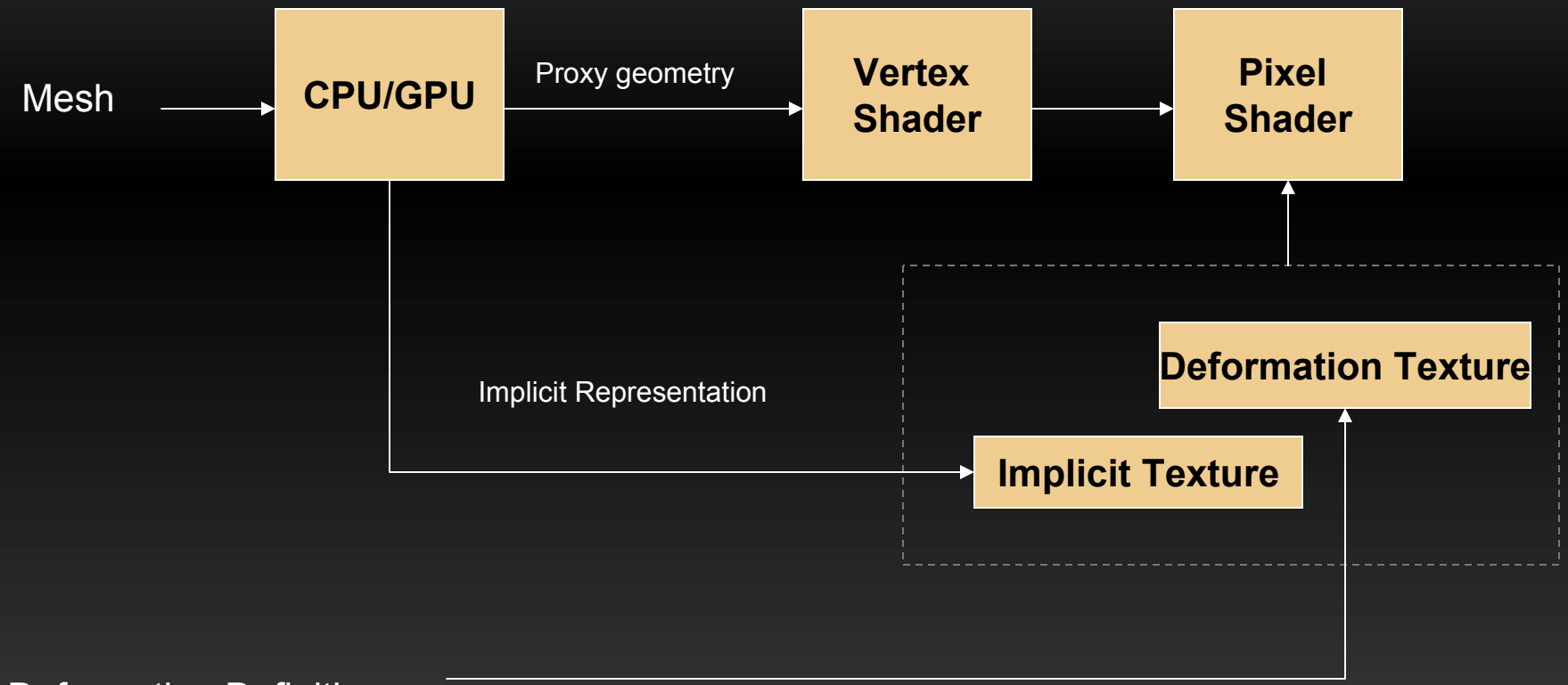
- No need for re-meshing
- Allows “virtual” modeling of solid and hollow objects

Deformation in the GPU

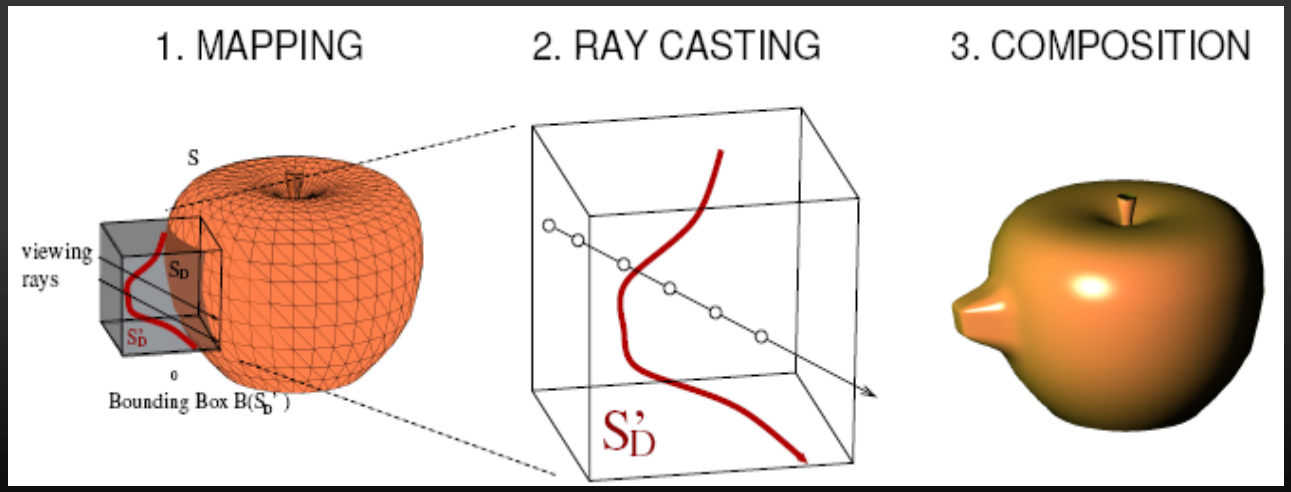




Our Approach



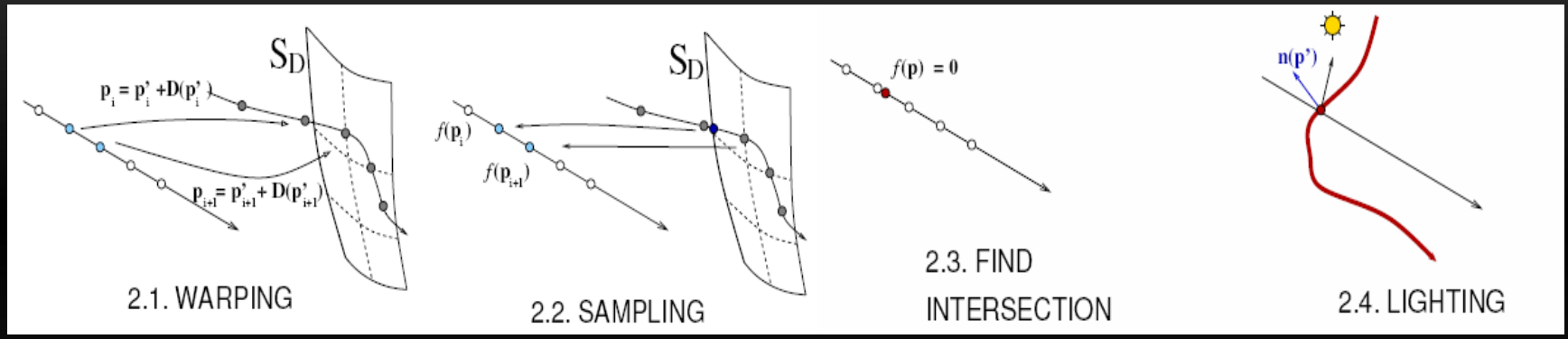
Pipeline



- Mapping: define a bounding volume B(S_D) where deformation is applied
- Render remaining part of mesh (orange) using traditional shaders
- Render B(S_D) using raycasting and *deformation shader*
- Composition: Allow a small overlap between B(S_D) and remaining mesh. Do alpha blending in the overlap region (the blending boundary is defined by the deformation texture, deformation must be zero in that boundary)

Pipeline

- Programmability of each stage helps realize a number of techniques.

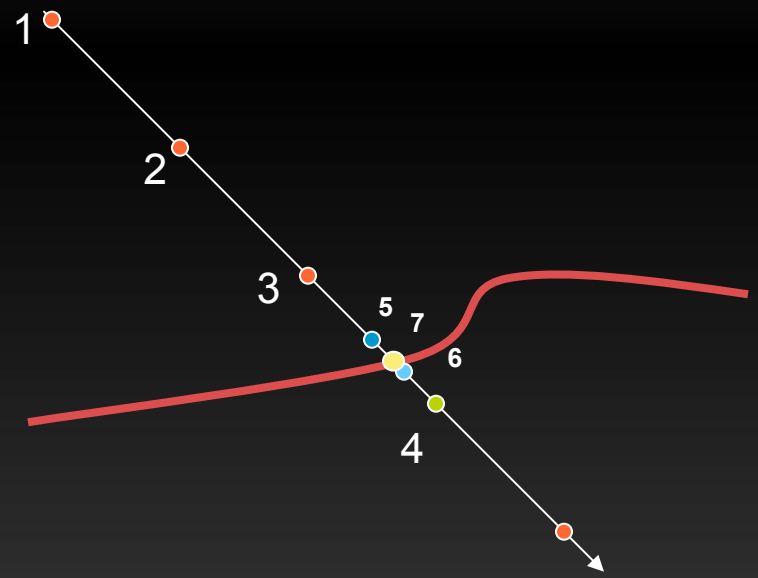


WARPING	SAMPLING	FIND INTERSECTION	LIGHTING
-Empirical model (procedural, FFD) -Displacement Map -Physics Simulation	Implicit Surface -Procedural -Distance Field -Sparse Implicit -Depth map	-Linear search -Linear search + binary refinement -Adaptive search (threshold) -LG surfaces	

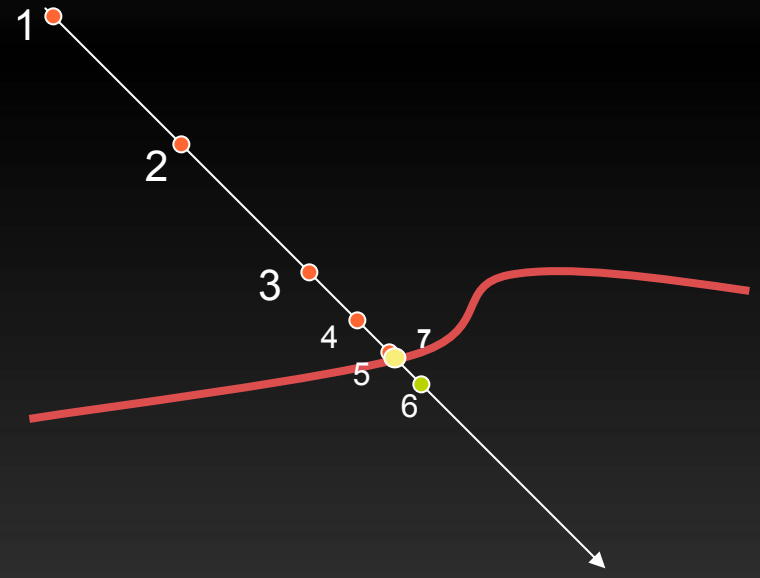
Ray traversal

- Rendering of an implicit function f is obtained by traversing a ray and finding the intersections with the zero-set:

$$f(\mathbf{p}) = 0$$



Linear search with binary refinement



Adaptive Linear search (threshold-based)

Ray traversal (2)

- For the deformed surface, raycasting must find the intersections with the zero-set:

$$f(\mathbf{p} + D(\mathbf{p})) = 0$$

  where *D* is a displacement texture (2D or 3D)

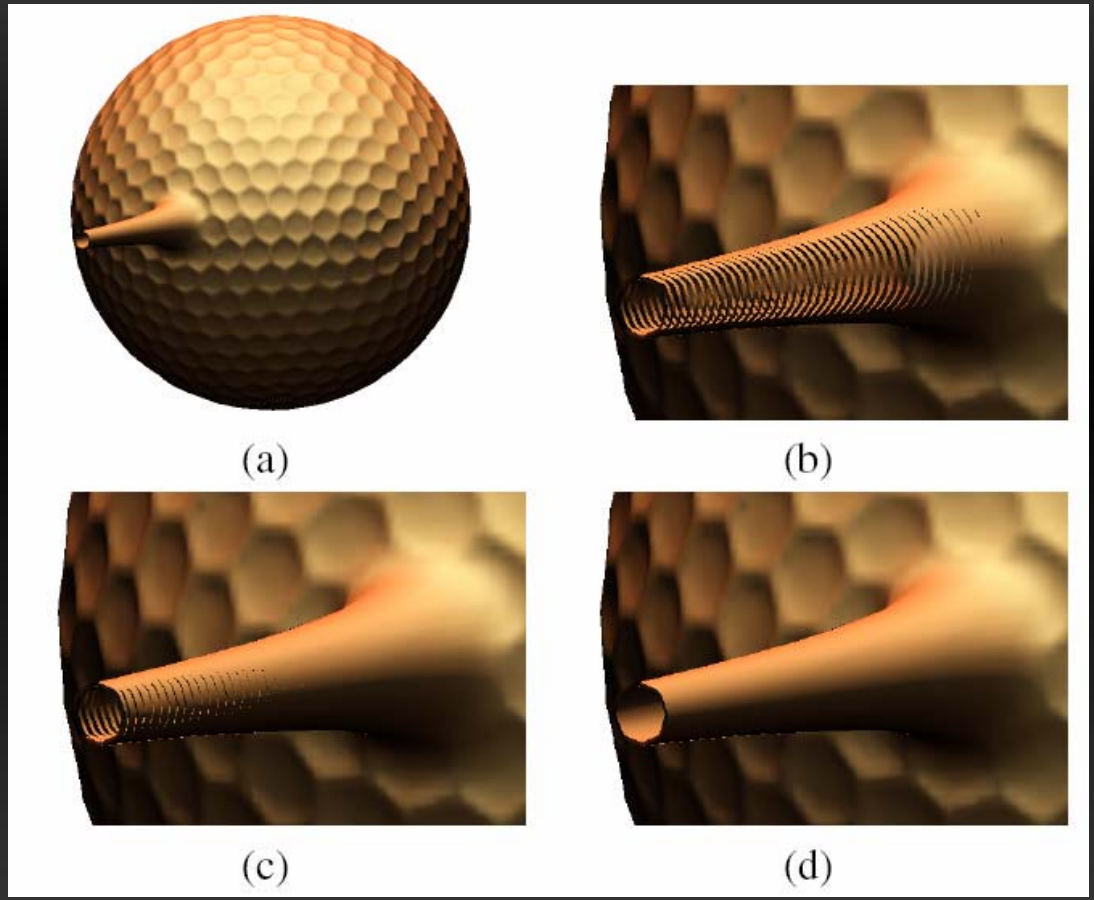
- Same search strategies should apply. Exception: Large deformations such as a long narrow pull → changes dramatically the spatial frequency of the mesh
  - Adaptive Sampling:

$$\delta t_i = \frac{1}{\left|(\mathbf{I} + \mathbf{J}_D(t))\mathbf{v}\right|} \delta s$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \delta t_i \, \mathbf{v}$$

Adaptive Sampling

- A long narrow pull applied to the golf-ball dataset (a)
- Linear search may miss some intersections (b)
- Threshold-based adaptation solves some problems, but threshold must be set dynamically to recover the deformed surface (c)
- Adaptive sampling using the displacement Jacobian finds the intersection (d)



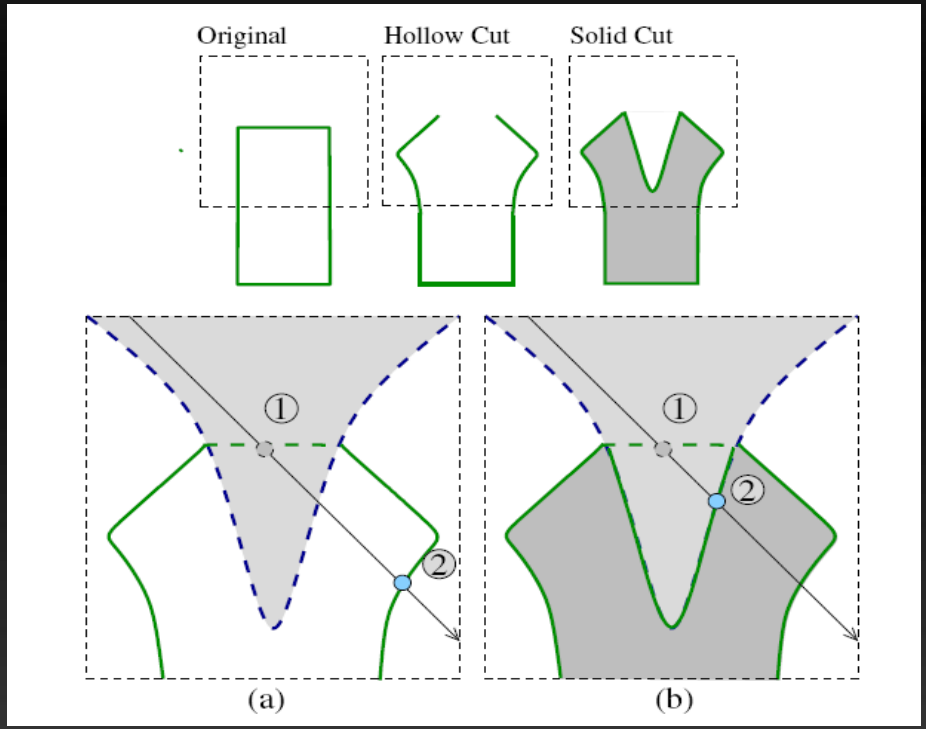
Lighting

- Whenever an intersection is found, we compute lighting parameters. Normal can be estimated from the original normal of the implicit representation (which in turn it's its gradient), and transformed using the Jacobian transpose of the deformation.

$$\vec{\mathbf{n}}'(\mathbf{p}) = normalize\left((\mathbf{I}+\mathbf{J}_D(\mathbf{p}))^{\top}\vec{\mathbf{n}}(\mathbf{p}+\mathbf{D}(\mathbf{p}))\right)$$

Cuts Shader

- One advantage of using deformation as a fragment shader, is the ability to define cuts without splitting a mesh.
- We use an alpha map to represent the cut geometry implicitly.
- We consider three cases:



Hollow:

- **Intersections are computed against the object representation only. Those intersections that fall within the cut area (e.g., (1)), are discarded**

Solid:

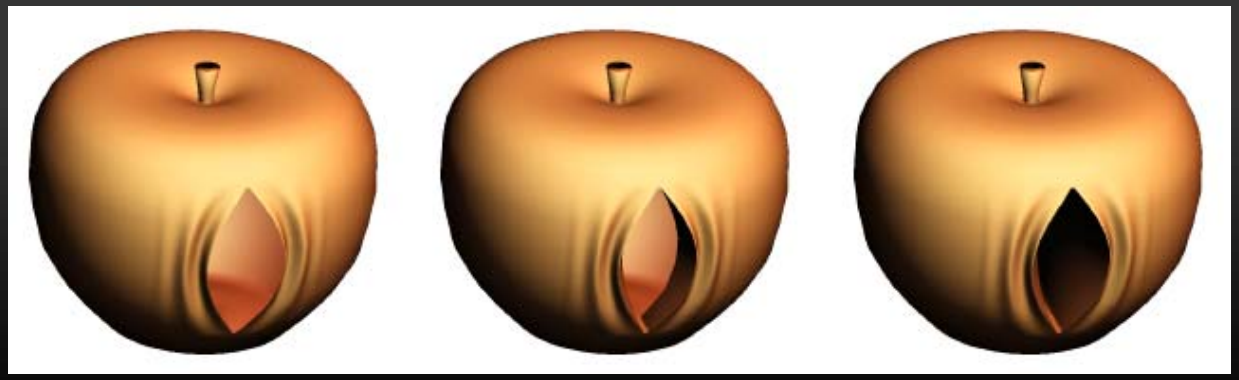
- **Intersections are computed against the object and the alpha map**

Thick Solid:

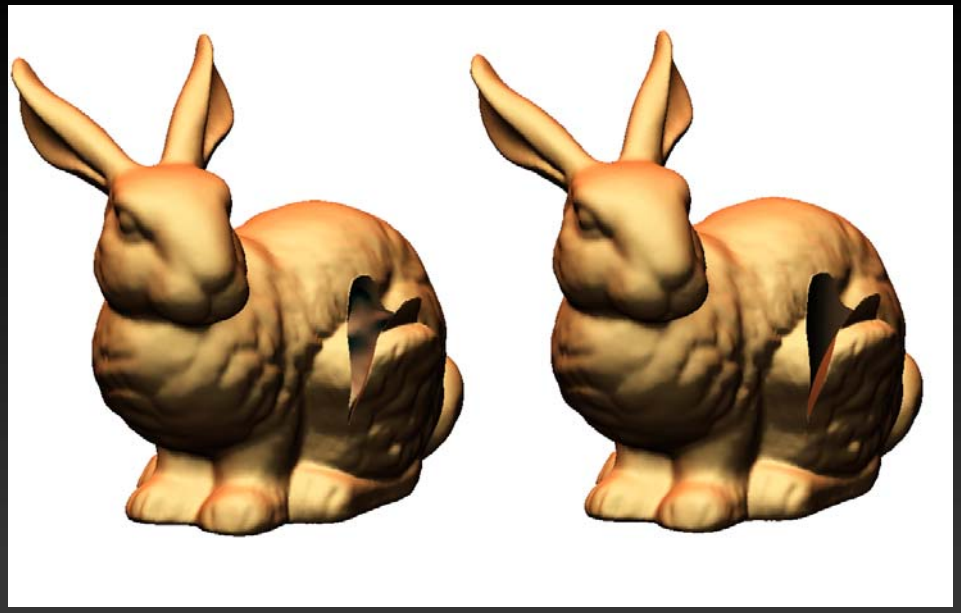
- **Special case of solid**

$$\hat{f}(\mathbf{p}) = \begin{cases} \tau - f(\mathbf{p}) & f(\mathbf{p}) > \frac{\tau}{2} \\ f(\mathbf{p}) & \text{otherwise} \end{cases}$$

Examples

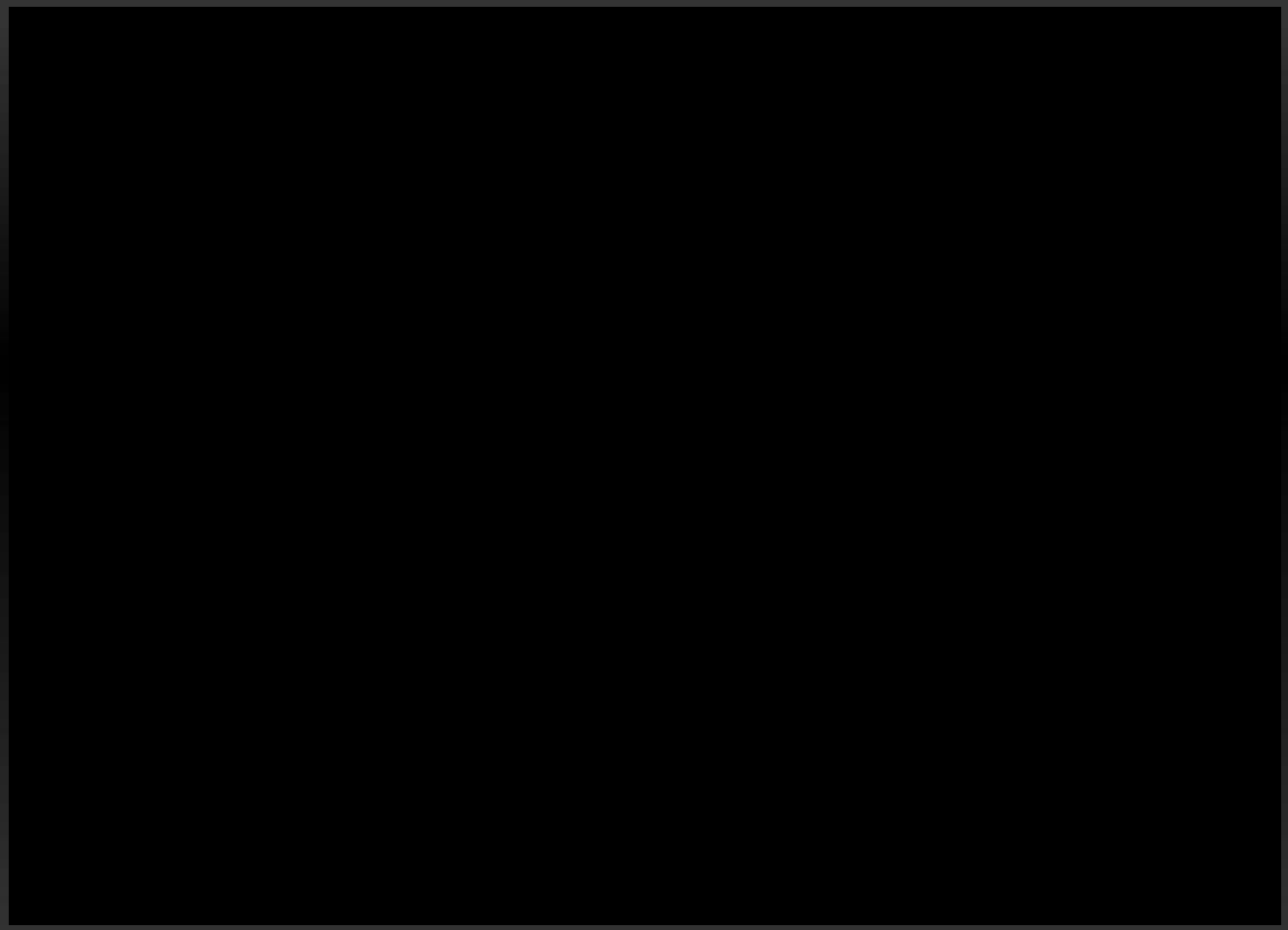


Hollow Thick-Hollow Solid



Hollow Solid

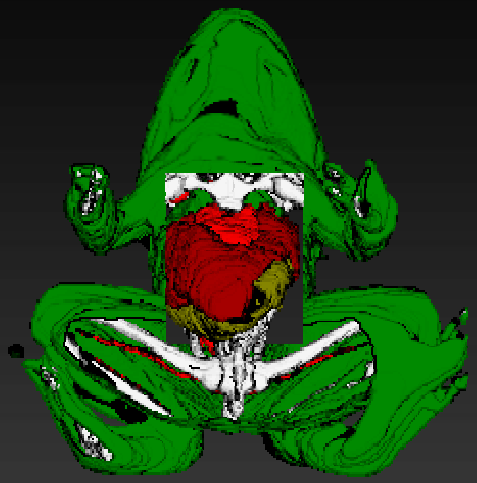
Video



Applications

- Education and Training
  - Surgery simulation
  - Virtual Dissection
- Games
  - Trauma Central

Trauma Center. Courtesy of Atlus



Virtual Frog Dissection

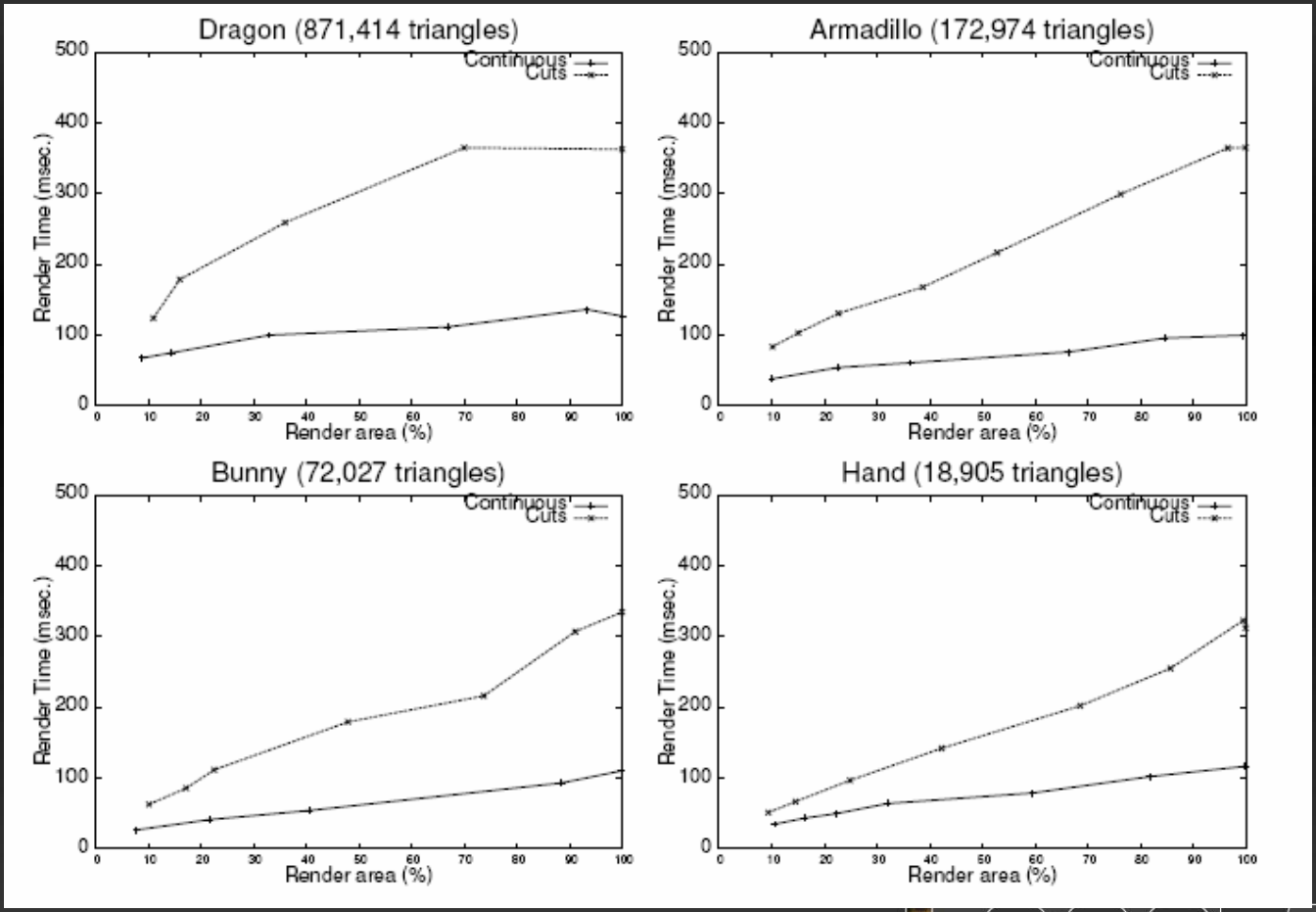
GPU Implementation

- Implemented as a single pass fragment shader
- Generation of implicit representation is done using a mix of CPU and GPU
  - It has been shown that it is possible to obtain signed distance fields in real-time
  - Less accurate representations are still useful, e.g., height map, cube depth map, which are very quick to obtain.

Quantitative Evaluation

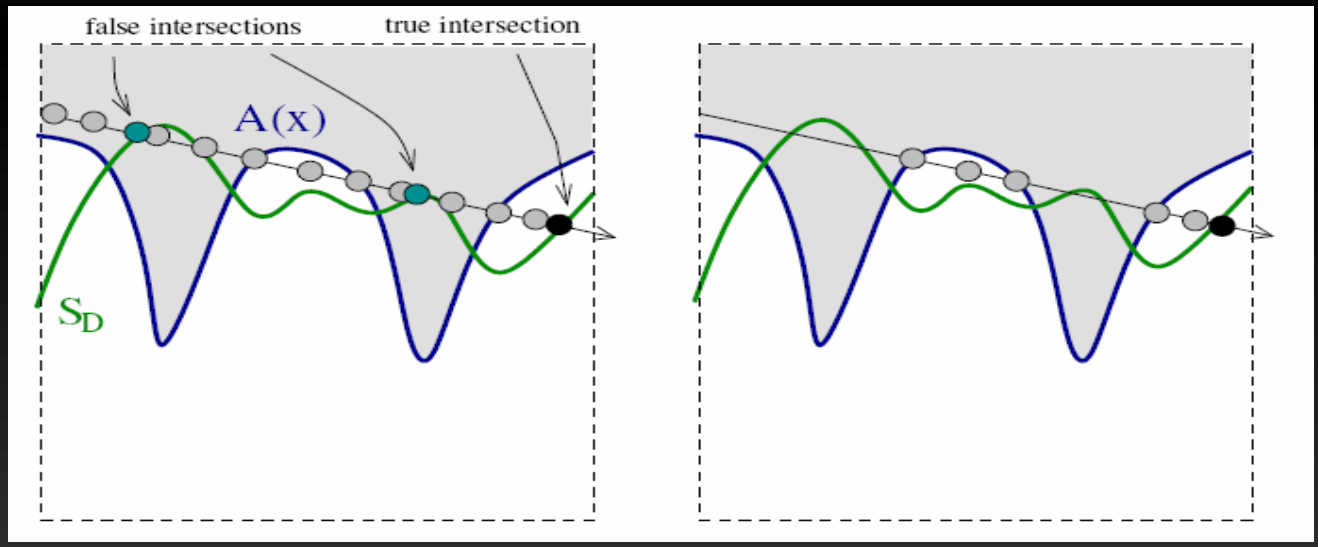
Comparison of continuous deformation shader vs. cut shader

Cut shader gets penalized due to “false” intersections



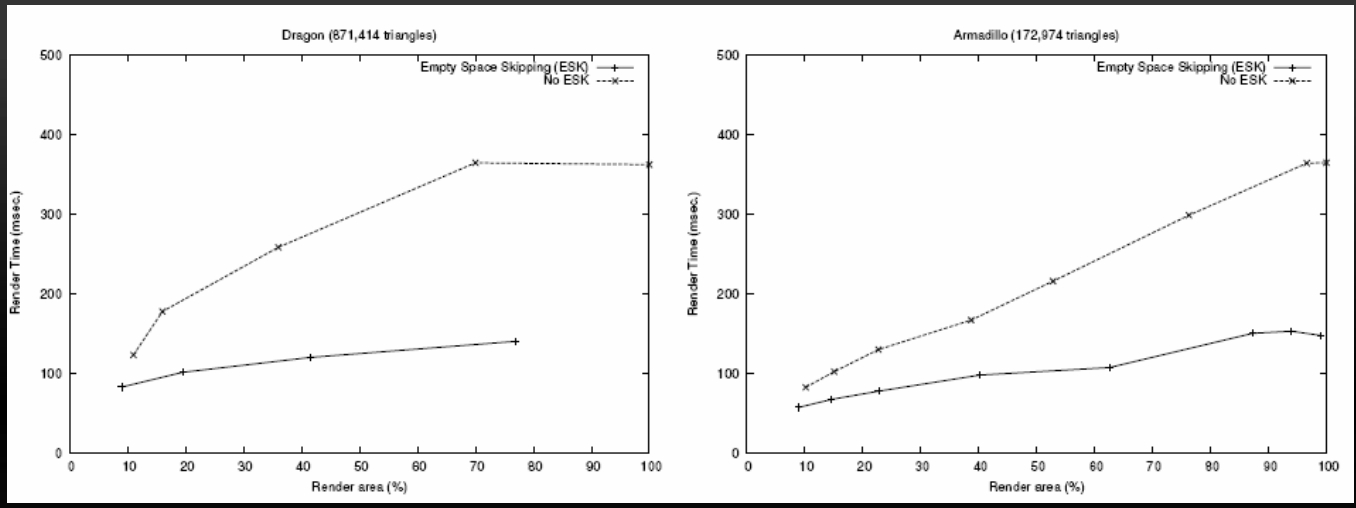
Empty Space Skipping

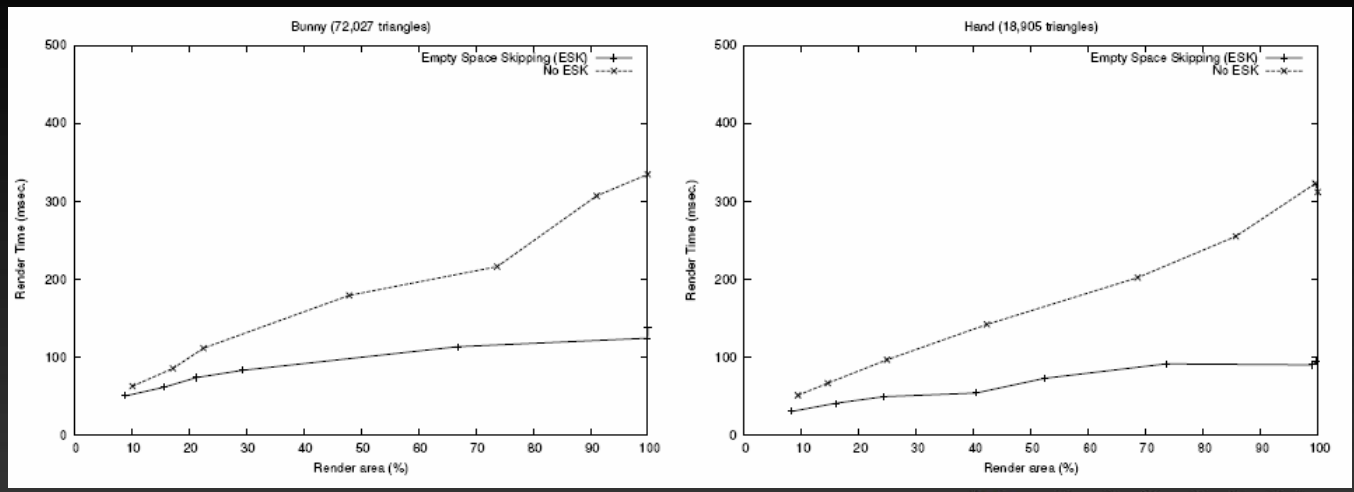
- To alleviate the cost of finding unnecessary intersections, the cut shader may exploit the implicit representation used for the cut to skip empty space.



Empty Space Skipping

Using ESK, the performance is as good as continuous deformation.





Discussion and Conclusion

- Improvement of vertex shaders/geometry shaders complement our approach: vertex shader for global/coarse deformation, fragment shader for local/fine deformation.
- Rendering cost dependent on image size, not on number of vertices. Constant speed for large meshes.
- Ray traversal is not necessarily coherent. Particularly for adaptive sampling, it can be costly. Coherent grid traversal can be exploited.
- Branching and early termination capabilities play critical role on acceleration techniques.
- Deformation can be simulated in the fragment shader. Changes the way we look at deformation → rendering problem

Thanks!

More info

http://www.caip.rutgers.edu/~cdcorrea/deforender