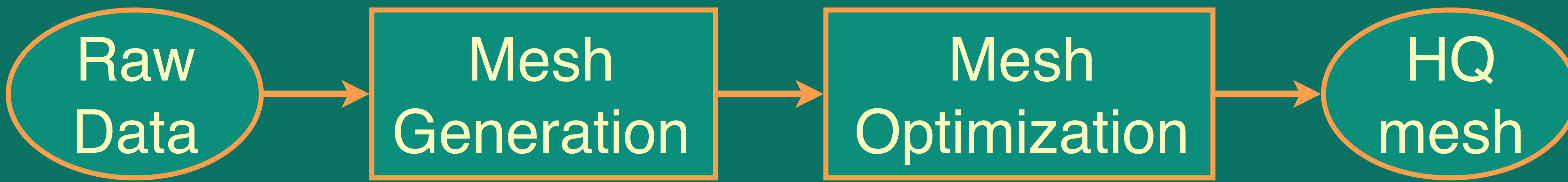
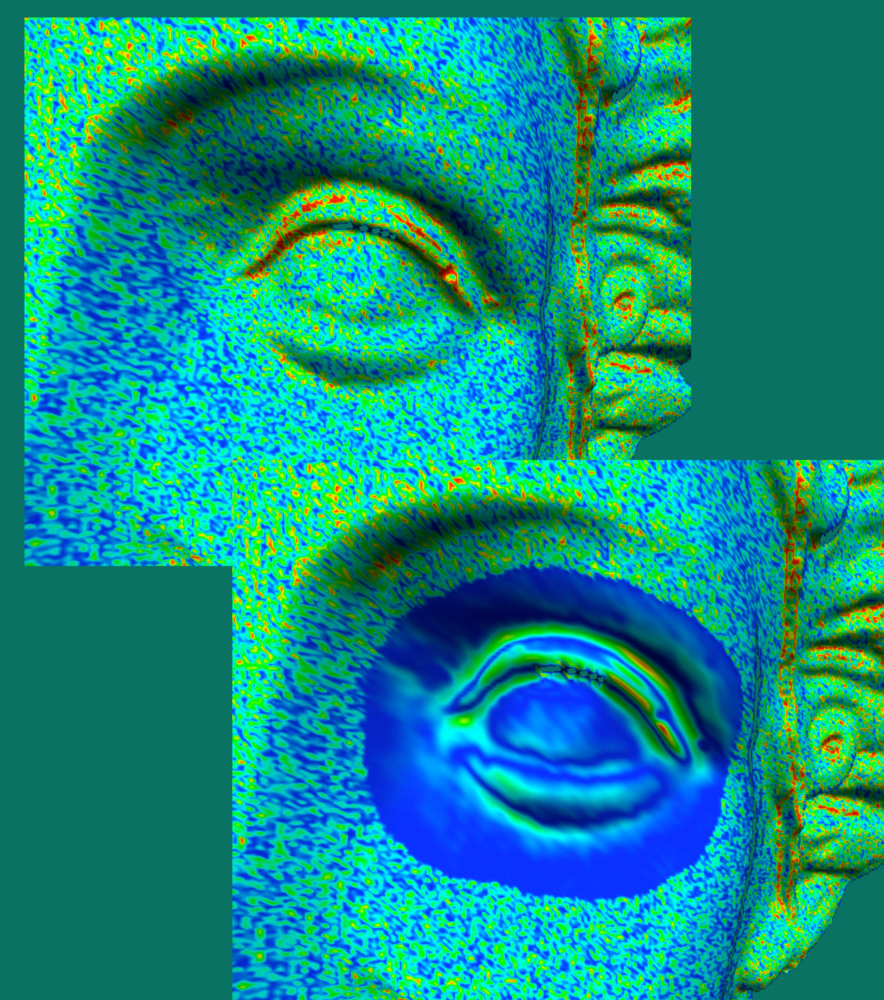
# GPU-based Tolerance Volumes for Mesh Processing

Mario Botsch, David Bommes, Christoph Vogel,
Leif Kobbelt


Raw Data
Mesh Generation
Mesh Optimization
HQ mesh

# GPU-based Tolerance Volumes for Mesh Processing

Raw Data → Mesh Generation → • Smoothing → HQ mesh



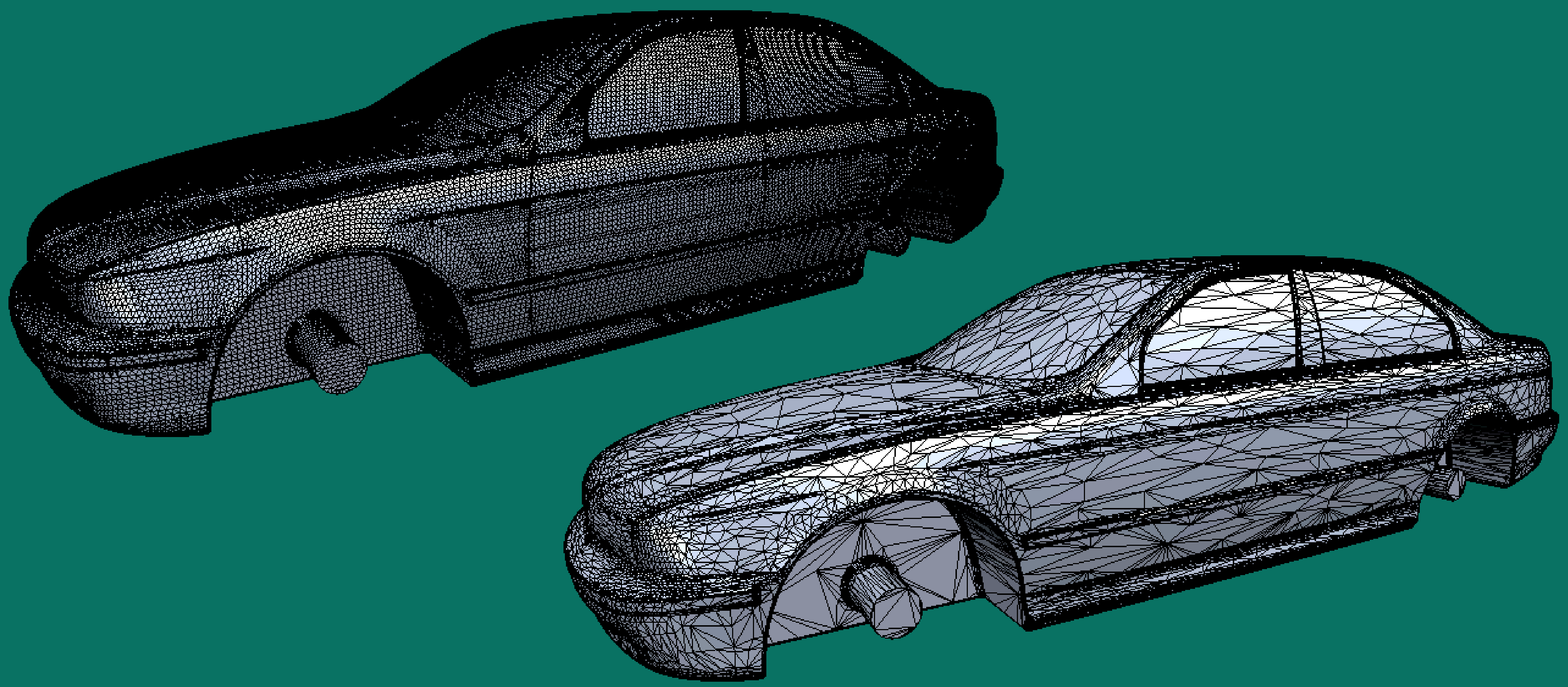


Computer Graphics Group
Mario Botsch

 **Mesh Processing**

Raw Data → Mesh Generation → 
- Smoothing
- Decimation

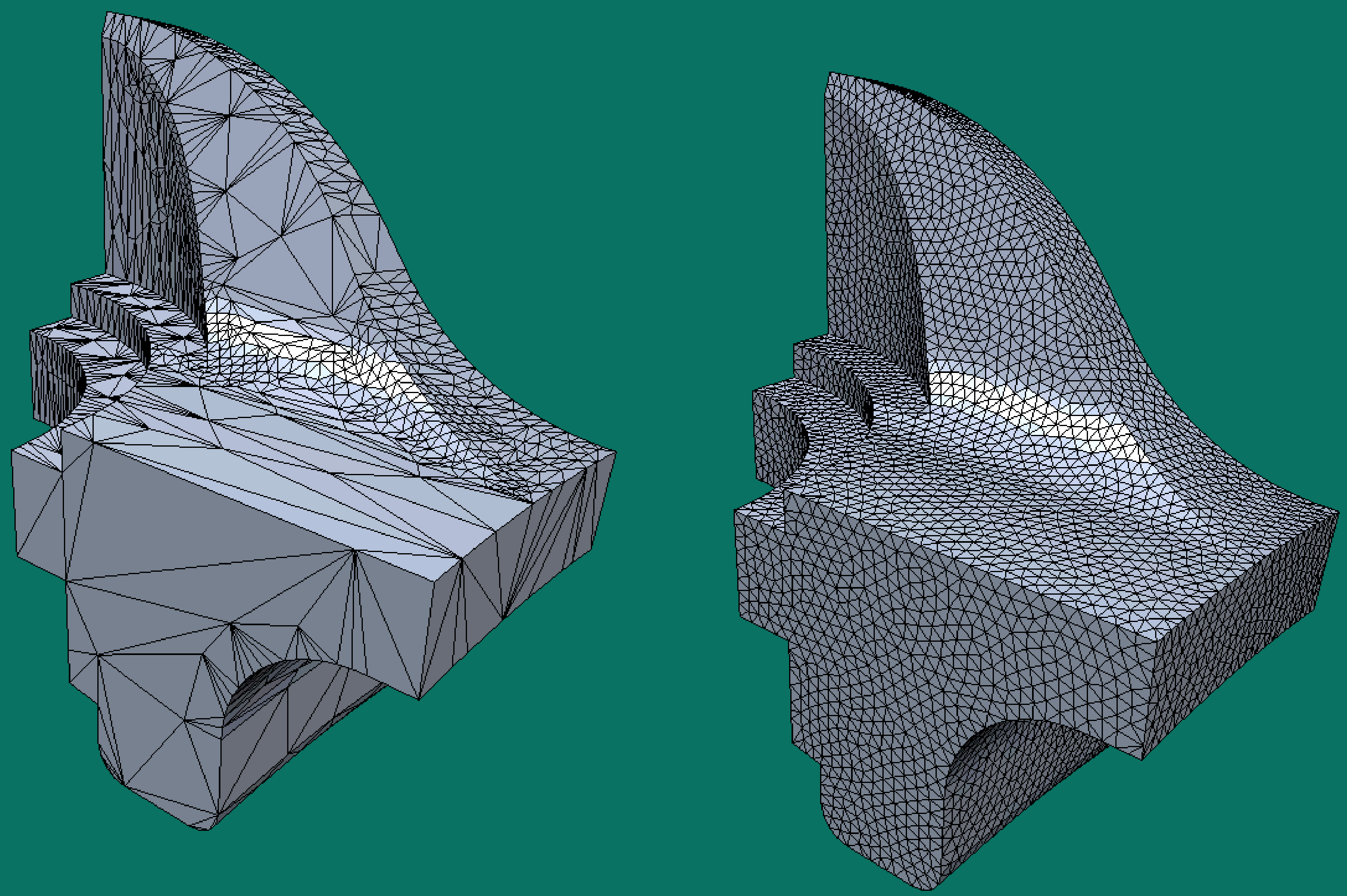
→ HQ mesh



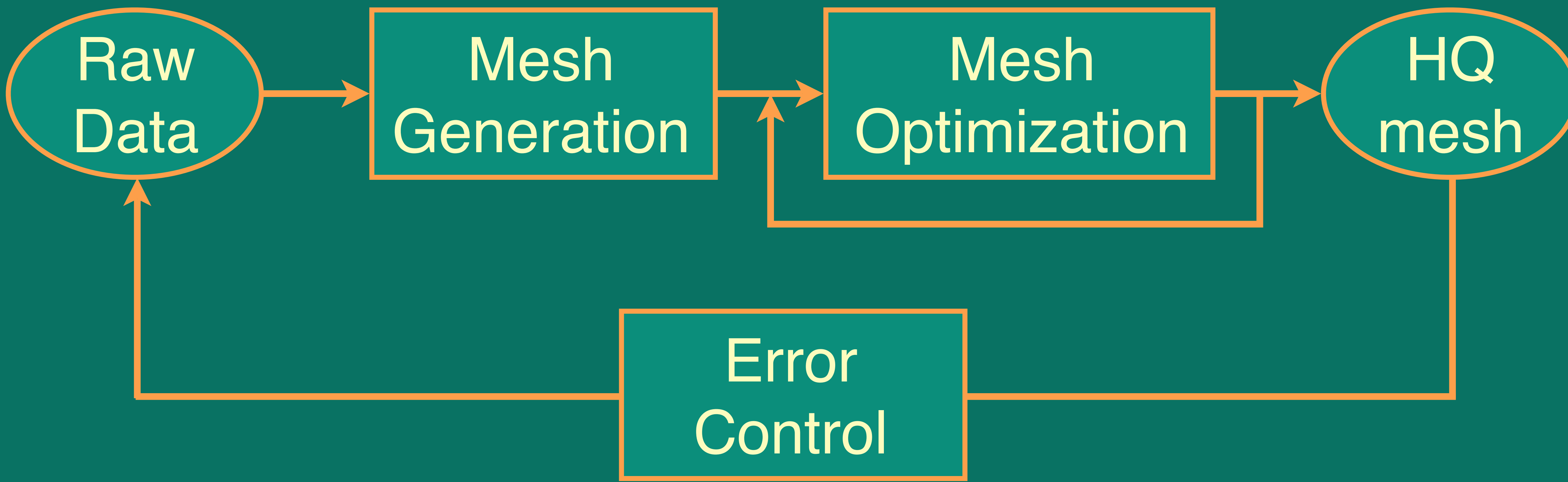
Raw Data → Mesh Generation → 

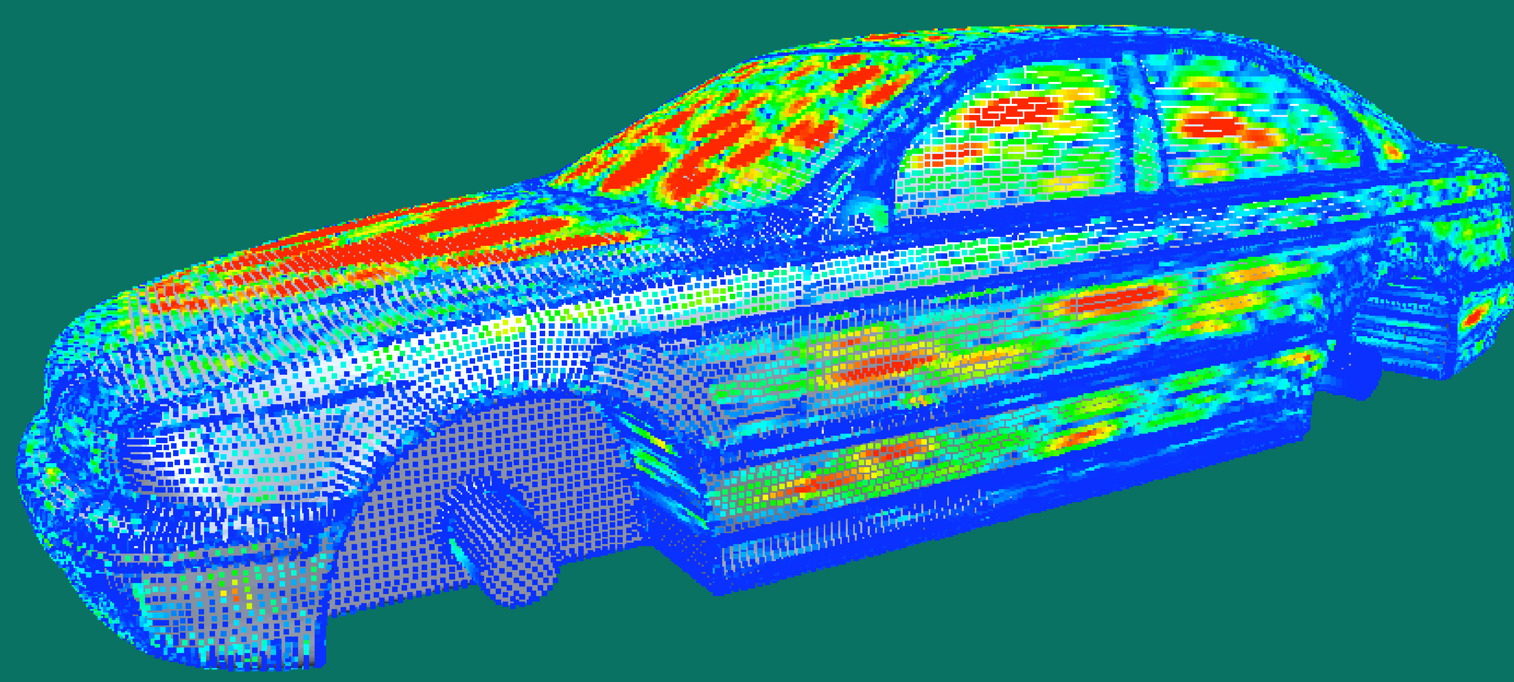
- Smoothing
- Decimation
- Remeshing

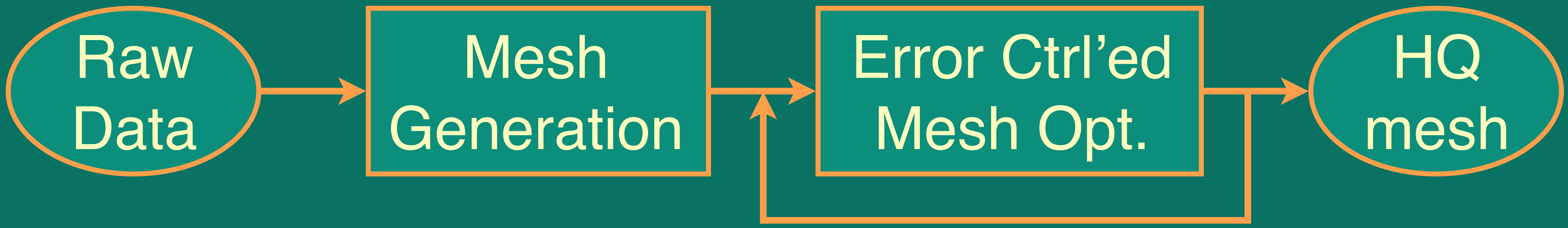
→ HQ mesh



# GPU-based Tolerance Volumes for Mesh Processing

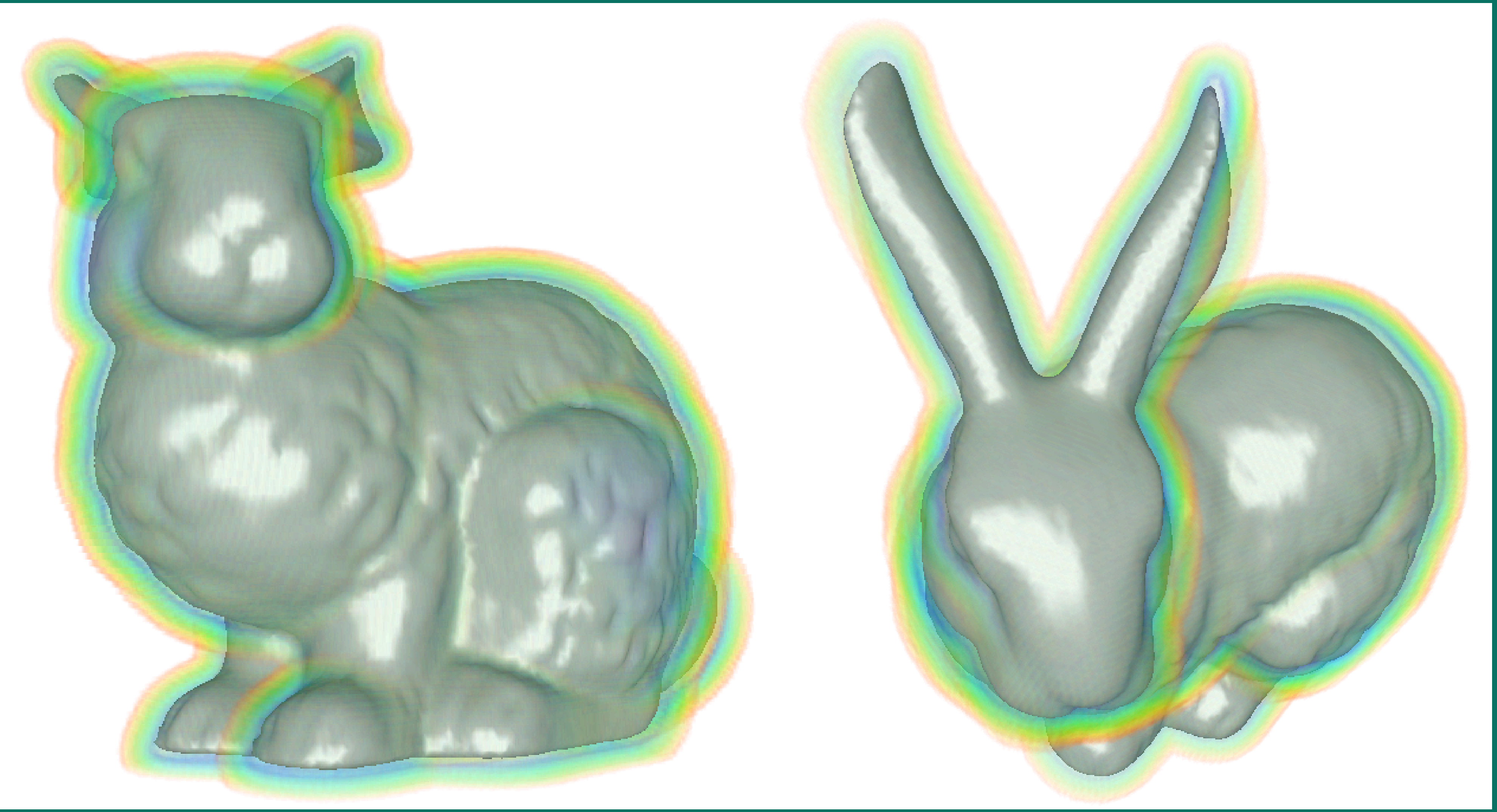



Raw Data
Mesh Generation
Error Ctrl'ed Mesh Opt.
HQ mesh

- Control global approximation error
  - Exact (or conservative)
- Each method may provide error control
  - Local errors may accumulate
- Need general global error control
  - Independent of mesh algorithm!

- Tolerance volume around original
  - Triangles have to stay within it



- General distance query
  - Implicit representation best suited (*distance = function evaluation*)
  - Approximate signed distance field
- Check modified triangle
  - Find SDF maximum over triangle
- How to approximate SDF ?

- Piecewise constant, $C^{-1}$, regular grid
  - Permission Grids [Zelinka & Garland]
  - Simple triangle test
  - High grid resolution
    (*low approx. order, alias artifacts*)

- Piecewise tri-linear, $C^0$, octree
  - Adaptively sampled DFs [Frisken et al.]
  - Low memory consumption
  - Complicated triangle test (*piecewise cubic function*)

- Piecewise linear, $C^{-1}$, BSP tree
  - Linear approx. SDFs [Wu & Kobbelt]
  - Low memory consumption
  - Complicated triangle test
    (*split triangles into BSP leaves*)

- Piecewise tri-linear, $C^0$, regular grid
  - Medium memory consumption (*regular grid, linear approximation*)
  - Complicated triangle test?

# GPU-based Tolerance Volumes for Mesh Processing

- Represent SDF as 3D texture
- Triangle test:  Just render it!
  - Automatic voxelization
  - Automatic tri-linear interpolation
- GPUs are efficient
  - Real-time *error control*
  - Real-time *error visualization*

# Overview

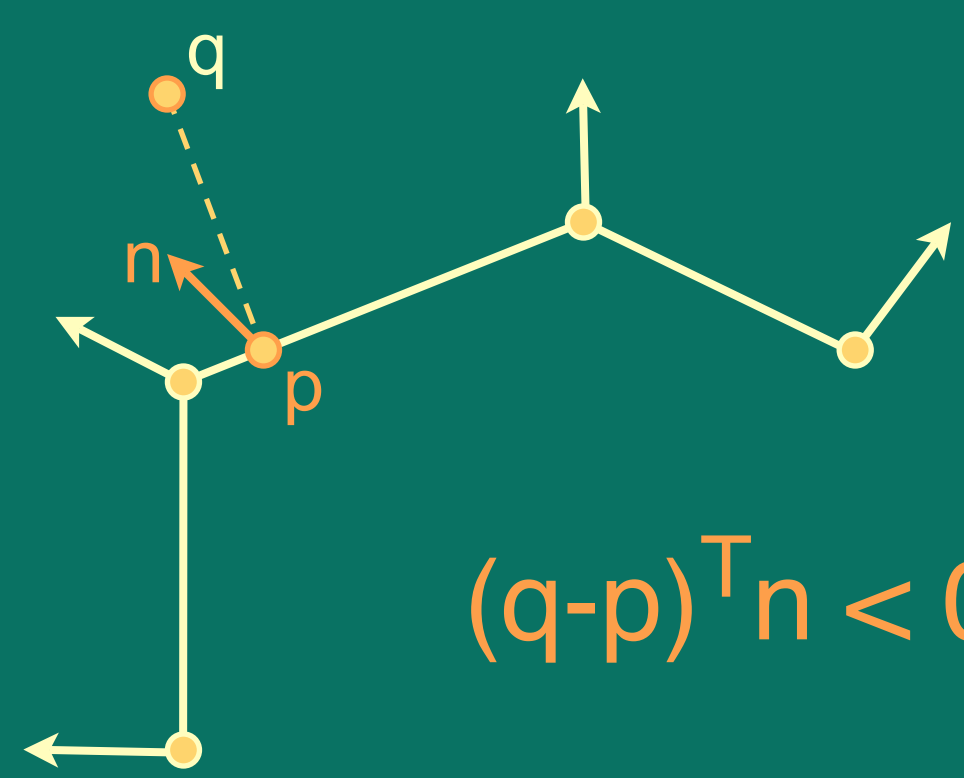
- Introduction
- Texture setup
- Triangle check
- Results

# SDF Texture

- Build regular grid $g_{ijk}$
- Compute distances $d_{ijk} = sdf(g_{ijk})$
- Convert to OpenGL 3D texture

# Distance Field Generation

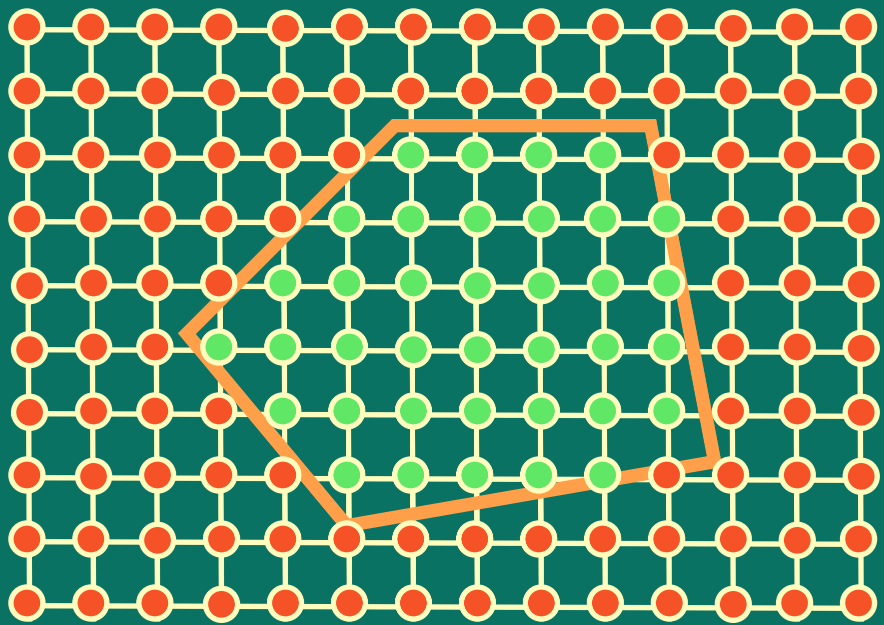
- Compute distance at each grid node
- Get *signed* distance from normal




$(q-p)^T n < 0$ ?

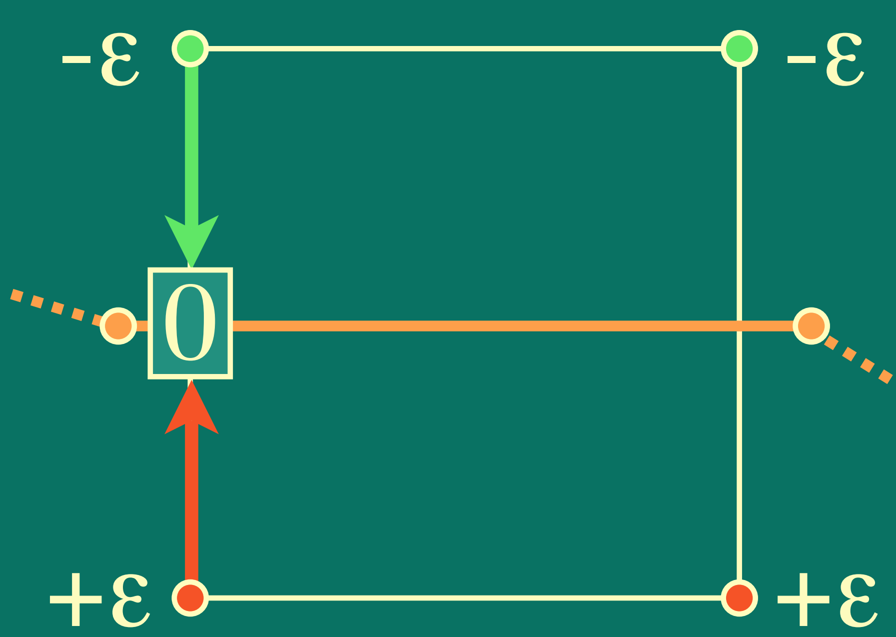
# Distance Field Generation

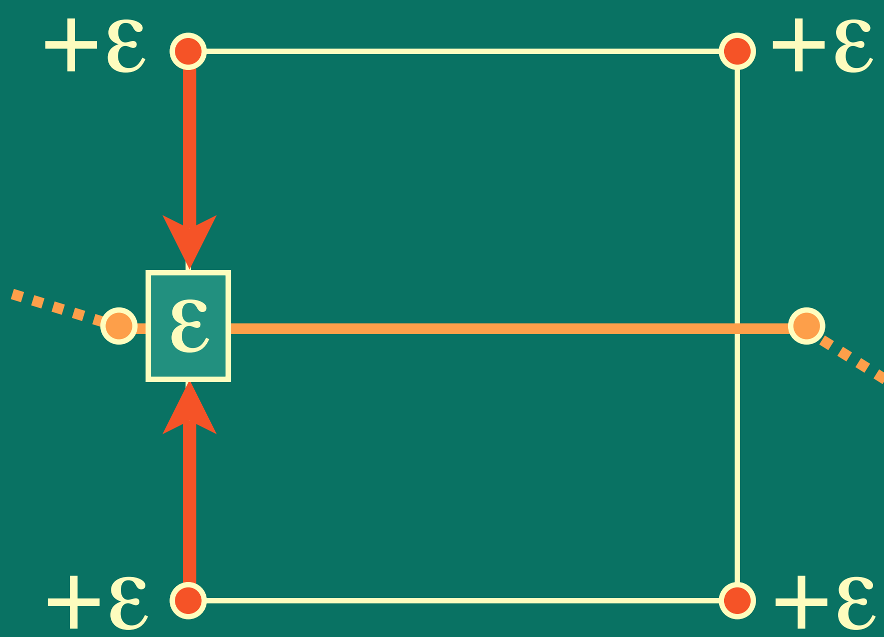
- Fast Marching Method
  - Initialization
  - March outward
  - March inward



# Non-closed Models?

- Use *unsigned* distance field instead
  - May over-estimate error by ½h



# SDF Approximation Error

- Piecewise tri-linear approximation
  - May under-estimate error by $\frac{\sqrt{3}}{2}h$
- Adjust user-specified tolerance

$$\varepsilon_{\max} \leftarrow \varepsilon_{\max} - \frac{\sqrt{3}}{2}h$$

# Texture Size

- Texture size should be power of 2
  - Fill up with empty rows/cols/slices
  - Waste of texture memory
- Improved by new extension
  - `ARB_texture_non_power_of_two`

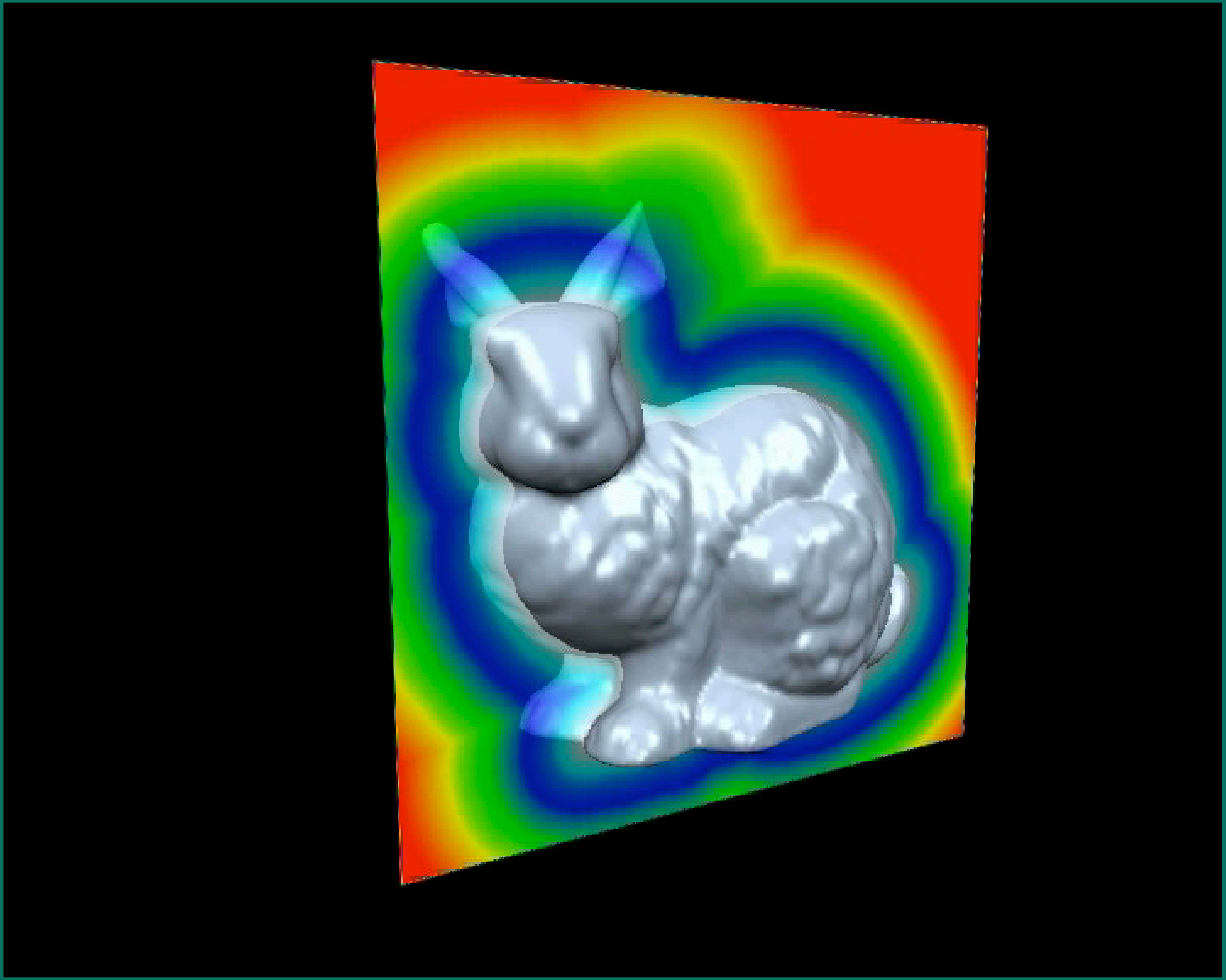
# Texture Value Type

- Use unsigned type `ALPHA8`
  - Map $[-\varepsilon_{\max}, +\varepsilon_{\max}] \rightarrow \{0, \ldots, 255\}$
  - 8 bit discretization of possible errors
  - Turned out sufficient
- Can also use
  - `ALPHA16`, `ALPHA32`
  - 32bit float (`ATI_texture_float`)

# Transfer Function

- Map interpolated texture value to
  - color (*error visualization*)
  - abs. distance value (*error check*)
- Apply after tri-linear texture interpolation
  - Texture shader or pixel shader
  - Post-classification

# Distance Volume

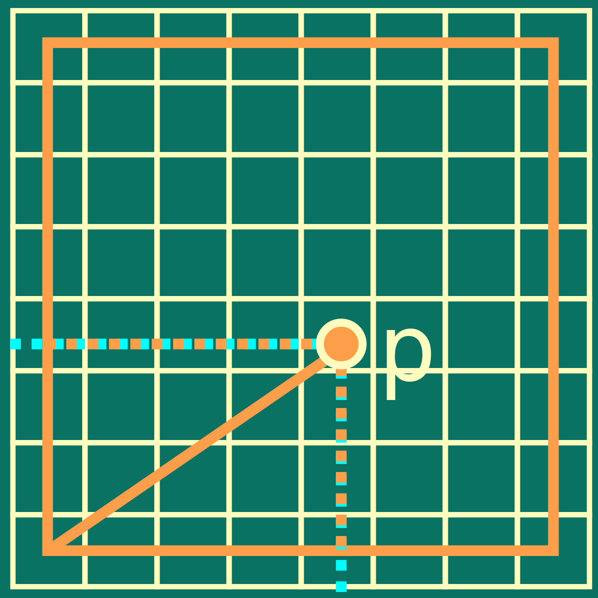


# Overview

- Introduction
- Texture setup
- Triangle check
- Results

# Texture Coordinates

- OpenGL assigns texture coordinates to texel *centers*, <u>not</u> *corners* !

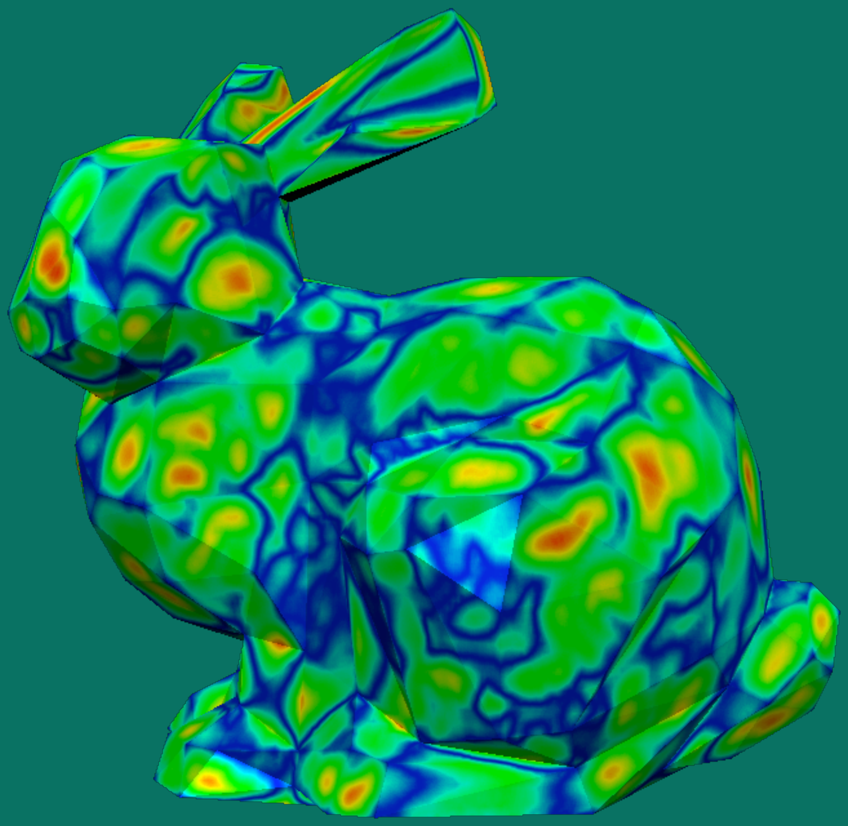



$$\mathbf{t}_x = \frac{\mathbf{p}_x - \mathbf{o}_x + h/2}{(\mathrm{res}_x + 1)h}$$

$$\mathbf{t}_y = \frac{\mathbf{p}_y - \mathbf{o}_y + h/2}{(\mathrm{res}_y + 1)h}$$

# Texture Coordinates

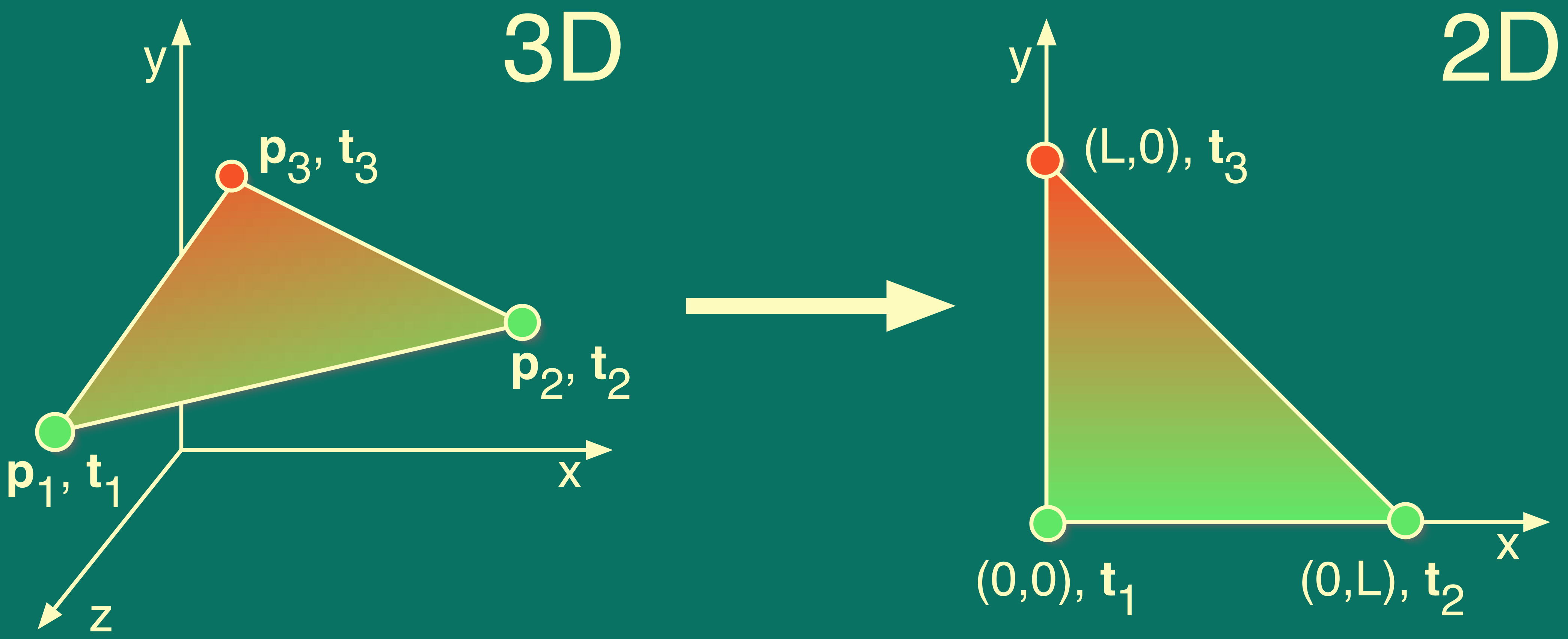
- OpenGL assigns texture coordinates to texel *centers*, <u>not</u> *corners* !
- Compute texture coords on GPU
  - Texture matrix
  - Vertex shader
- Real-time visualization
  - 15M dynamic triangles



# Triangle Checking

- How to choose viewing position for error checking?
  - Each triangle has to be fully visible
  - Generate sufficiently many pixels
- 3D coordinates not important, only texture coordinates matter

# 2D Positions, 3D Tex. Coords



# Triangle Check

- Draw 2D triangle with 3D texcoords
- Use alpha transfer function

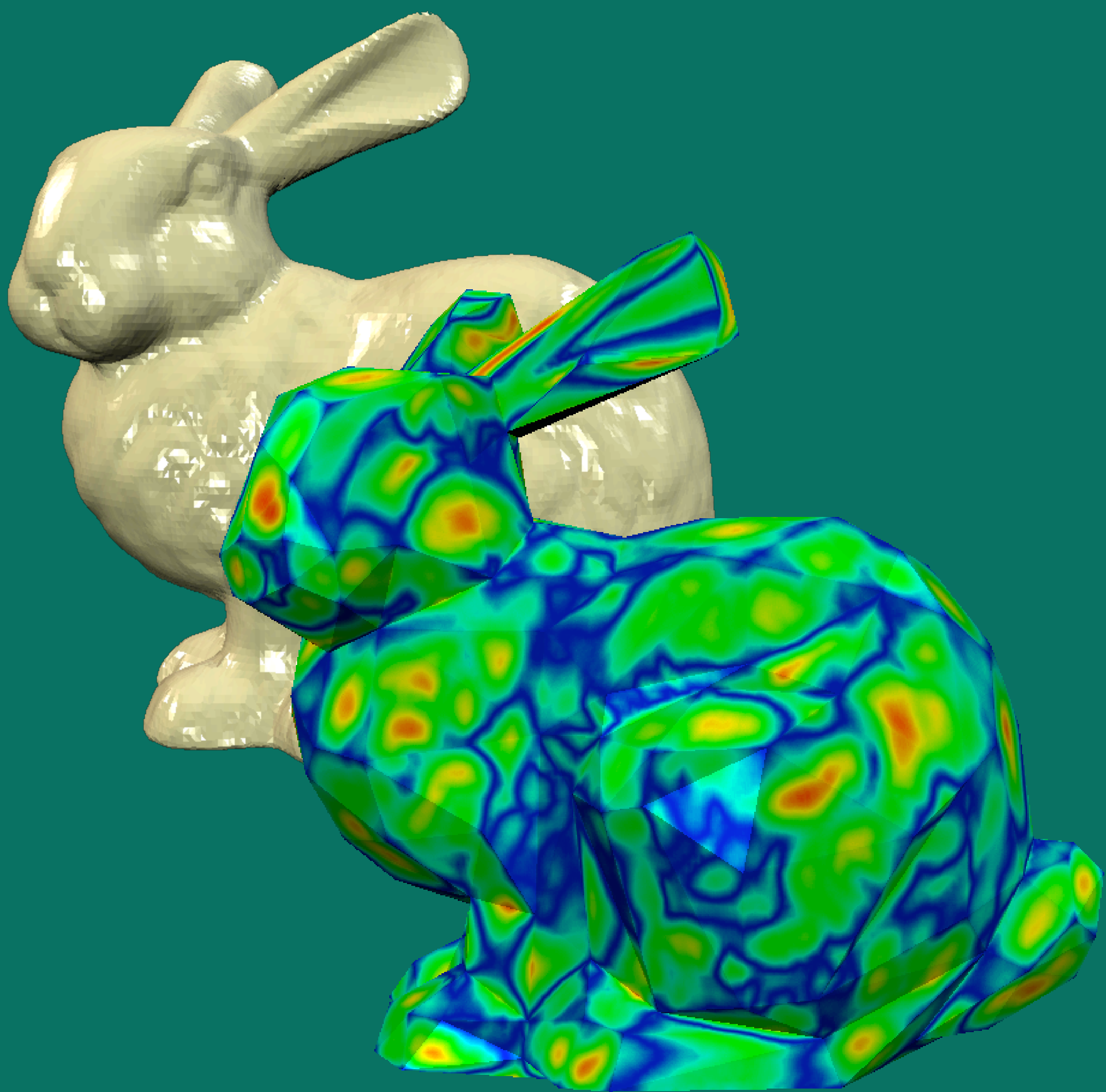
$$\alpha(x) = \begin{cases} 0, & x \leq \varepsilon_{\max} \\ 1, & \text{otherwise} \end{cases}$$

- Only invalid pixels will be drawn
  - Pixel drawn $\Rightarrow$ triangle invalid
  - `ARB_occlusion_query`

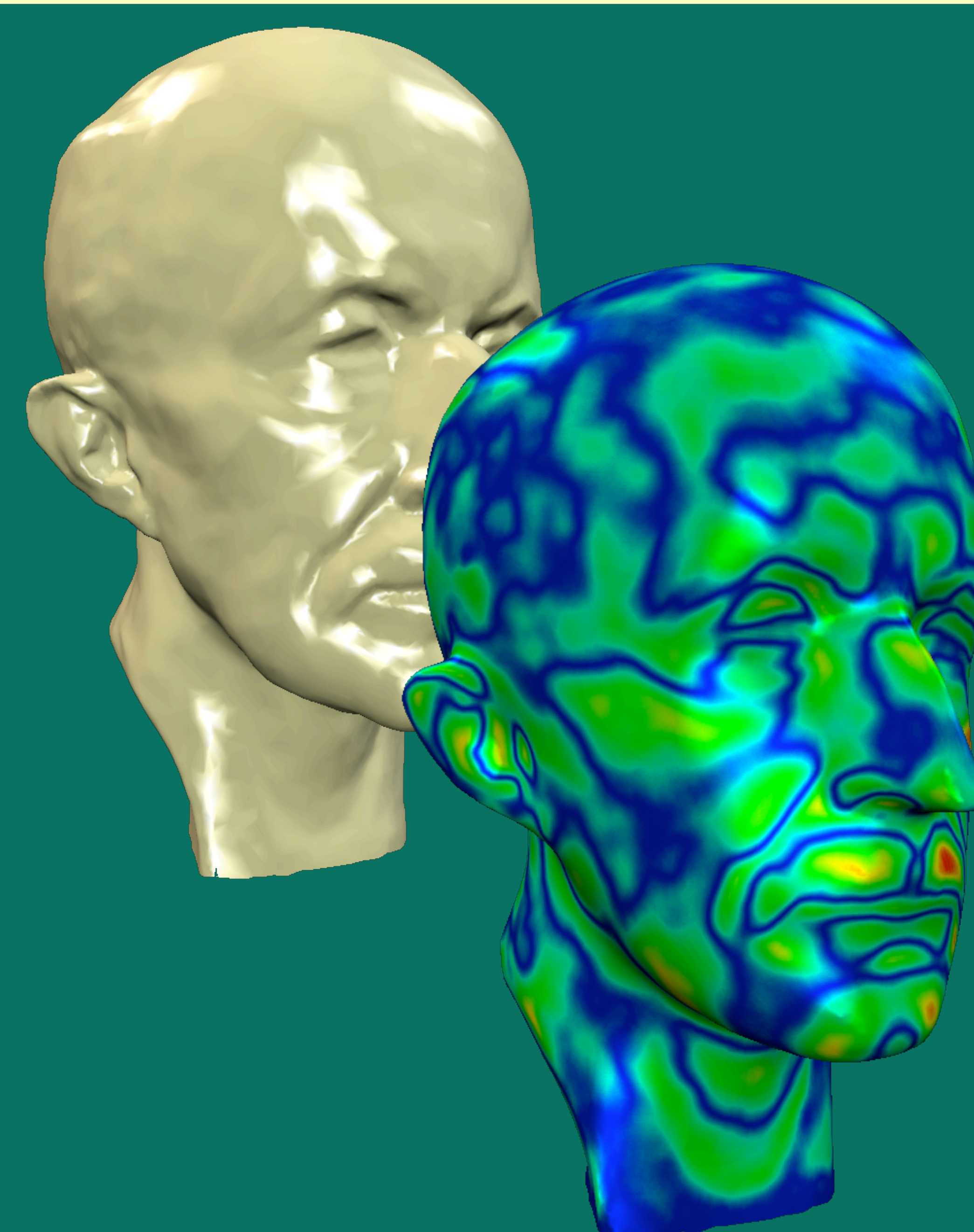
# Overview

- Introduction
- Texture setup
- Triangle check
- Results

# Error Control & Visualization



Decimation



Smoothing

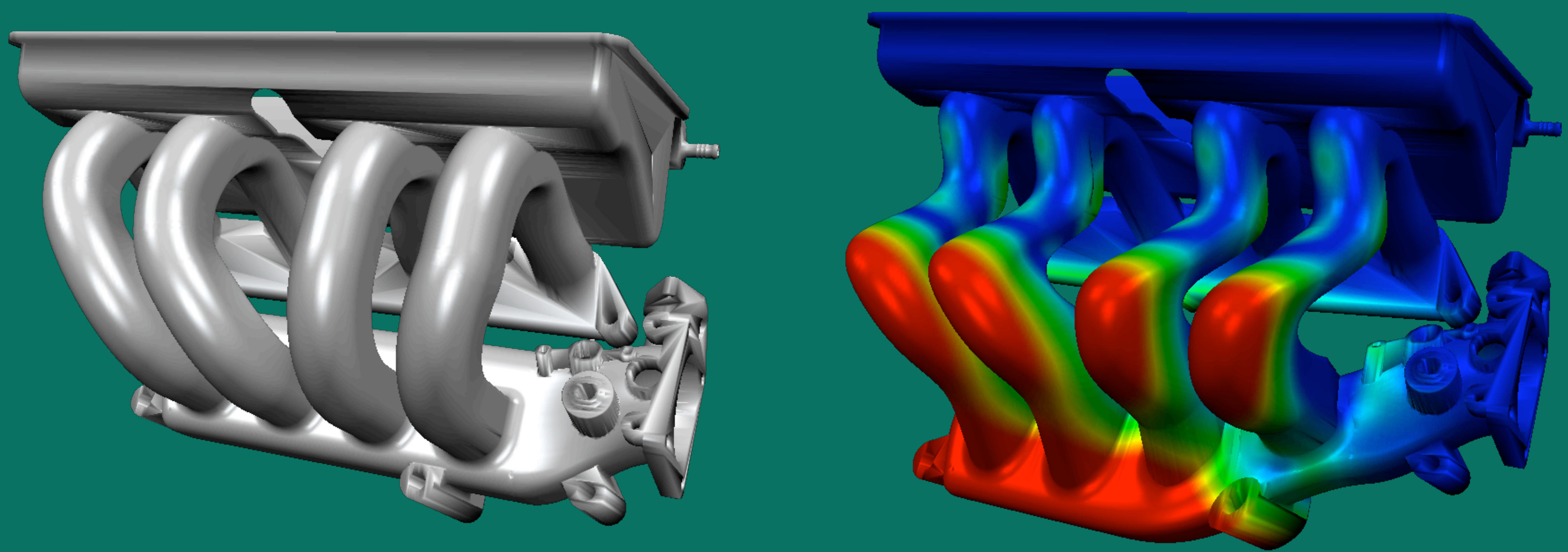
# Decimation Timings

- GPU decimation is faster than
  - Hausdorff-based decimation (2×)
  - Permission grids (1.5× - 2×)

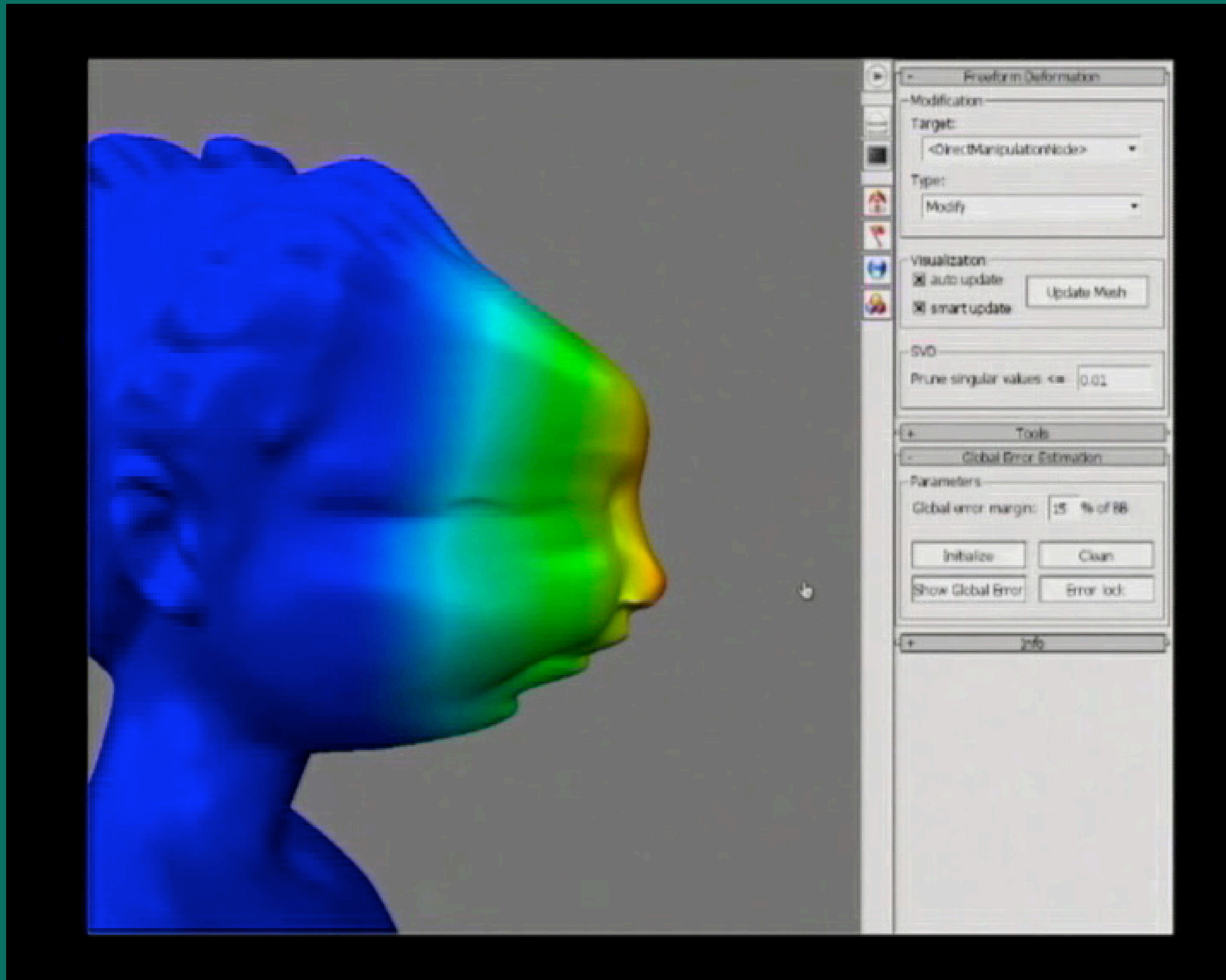
| Model | Input | Grid | FM | Deci | QEM |
|---|---|---|---|---|---|
| Bunny | 70k | 74×73×58 | 1.7s | 2.3s | 2.0s |
| Horse | 96k | 85×71×42 | 1.9s | 3.2s | 2.6s |
| Venus | 269k | 53×74×74 | 5.0s | 10.3s | 8.8s |
| Buddha | 1M | 45×101×45 | 18.0s | 39.6s | 34.5s |

# Freeform Deformation

- Exact deviations due to error control
- Real-time visual feedback



# Freeform Deformation



# GPU-based Tolerance Volumes

- General global error framework
  - Independent of application
- Simple implementation
  - Complicated tasks done by GPU
- Efficient GPU implementation
  - Error check: 3M tri/sec
  - Error visualization: 15M tri/sec

# Acknowledgments

- David Bommes (Framework)
- Christoph Vogel (Fast Marching)
- Michael Lambertz (3D Texture)
- Andreas Wiratanaya (FFD)
- Benjamin Molitor (Vol. Vis.)