# Level of Detail

# Level of Detail (LOD)

Allows for more efficient memory and processing based on resolution needed for good user experience

- e.g. farther away things should be inexpensive

# LODs

Mipmaps

Antialiasing

**Mesh Reduction**

**Billboards**

# Mesh Reduction

Closer objects map to more pixels, so require higher resolution models

Distant models map to fewer pixels, so lower resolution models will work

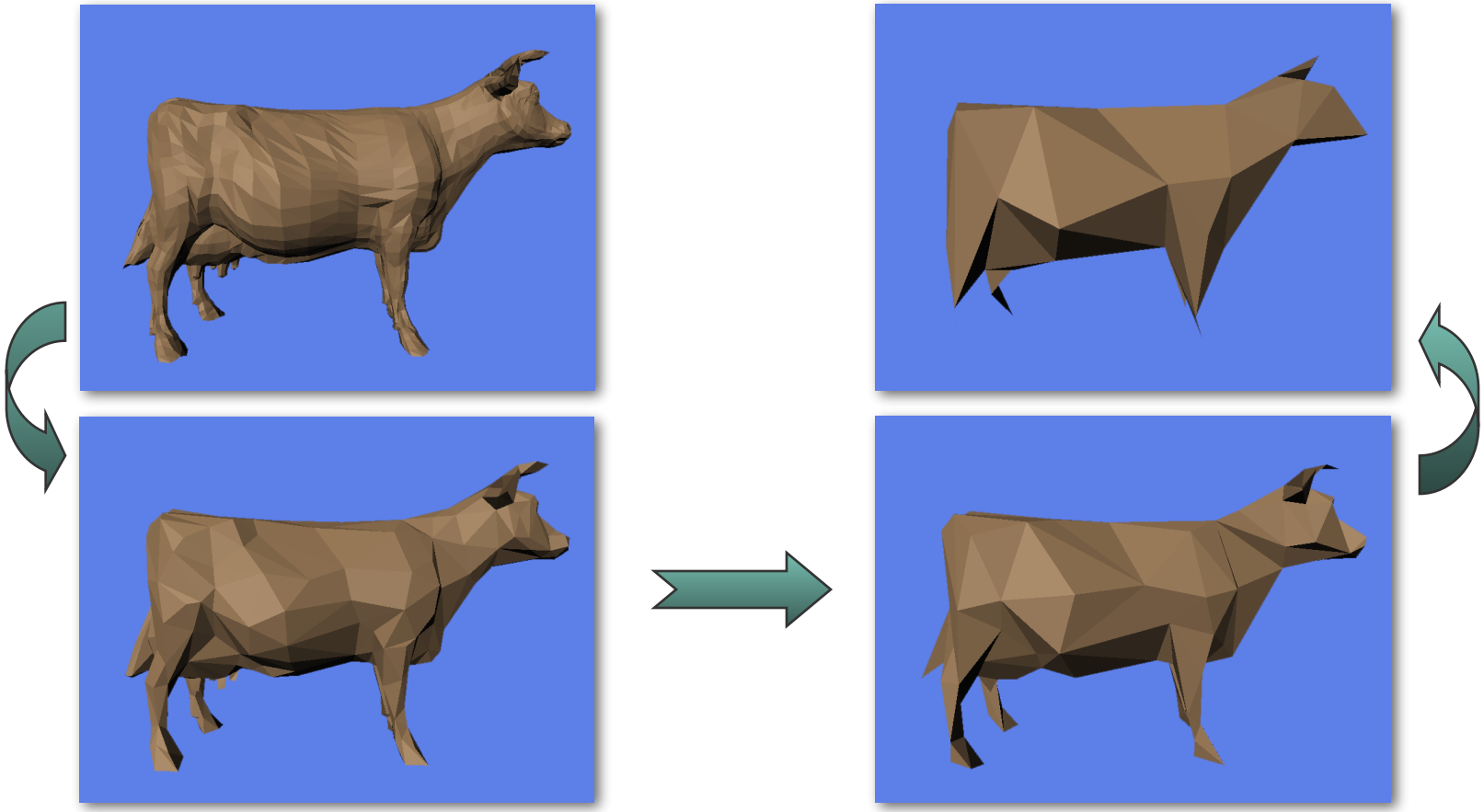
# Standard LOD

Create finite set of models

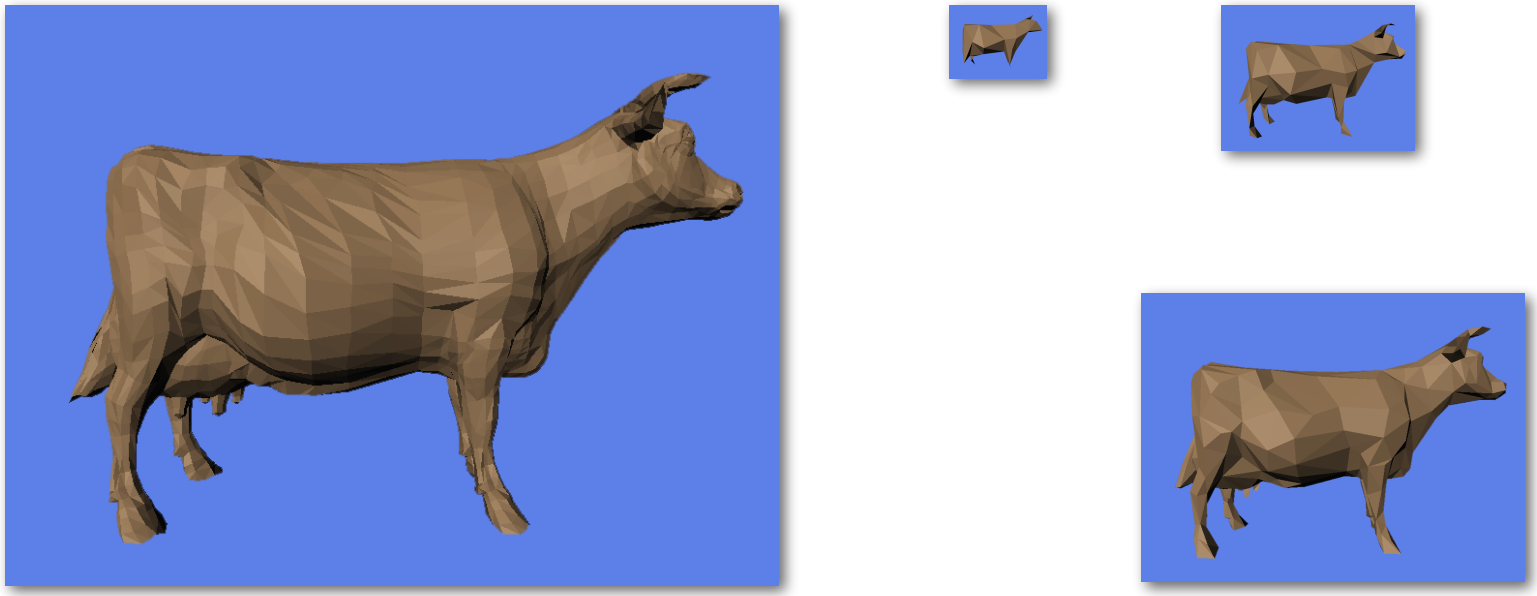
- Typically aim for models with n, n/2, n/4, … polygons
- Models hand-generated or automatically decimated

Switch models based on the distance to viewer

# LOD Example



# LOD Example



# Mesh Decimation

Goal: Reduce mesh complexity (eliminate triangles) while maintaining “good” approximation

- Error metric evaluates progress at each step

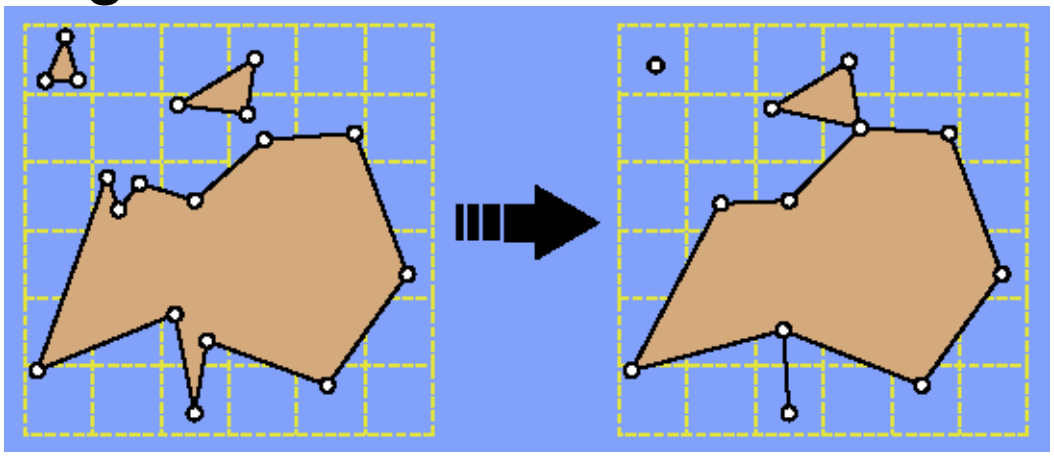
# Vertex Clustering

Partition space into cells

- Grids [Rossignac-Borrel], spheres [Low-Tan], octrees, etc

Merge vertices within same cell

- Will degenerate

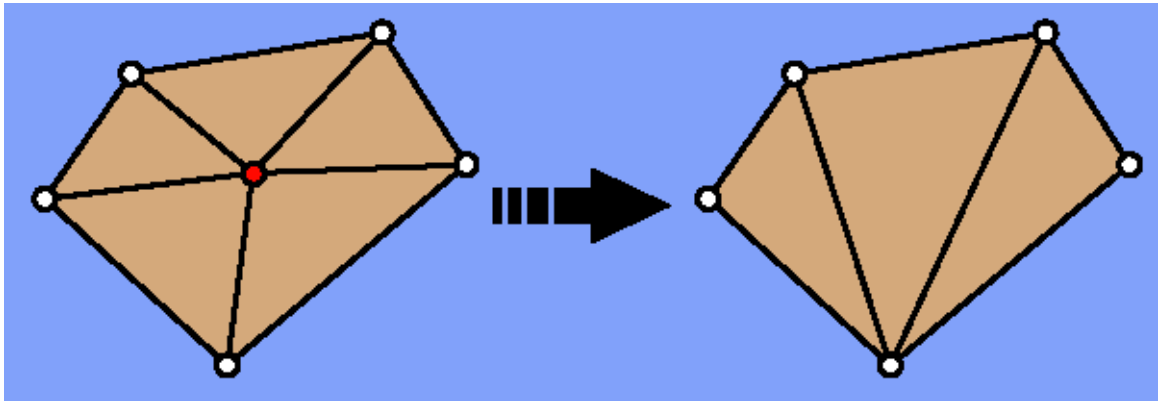


(Michael Garland, http://graphics.cs.uiuc.edu/~garland)

# Vertex Decimation

On original model, iteratively:

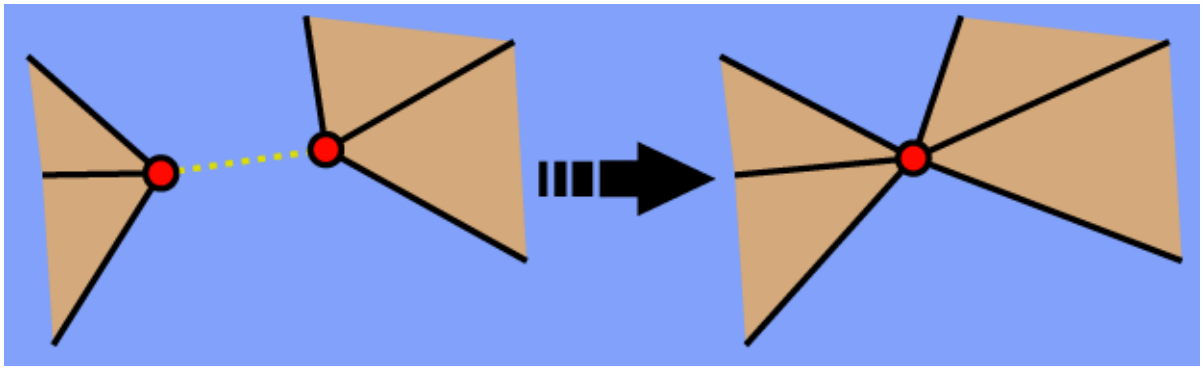
1. Rank vertices according to importance
2. Remove unimportant vertex and re-triangulate



(Michael Garland, http://graphics.cs.uiuc.edu/~garland)

# Vertex Pair Contraction

Contract any pair of vertices to achieve topological simplification

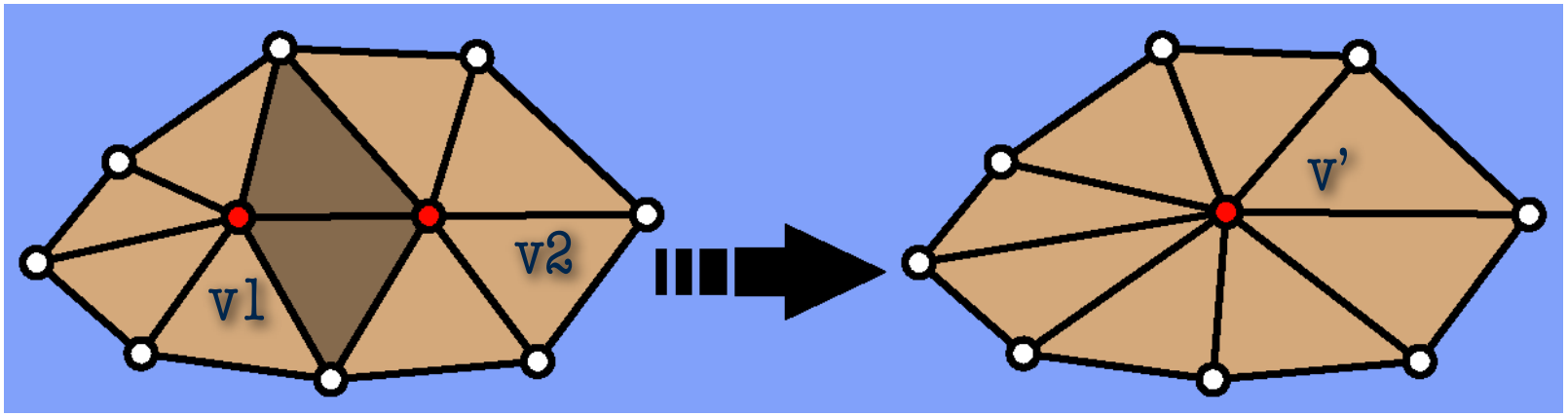


(Michael Garland, http://graphics.cs.uiuc.edu/~garland)

# Edge Contraction

Single edge contraction (v1,v2) → v'

1. Move v1 and v2 to position v'
2. Replace all occurrences of v2 with v1
3. Remove v2 and all degenerate triangles




(Michael Garland, http://graphics.cs.uiuc.edu/~garland)

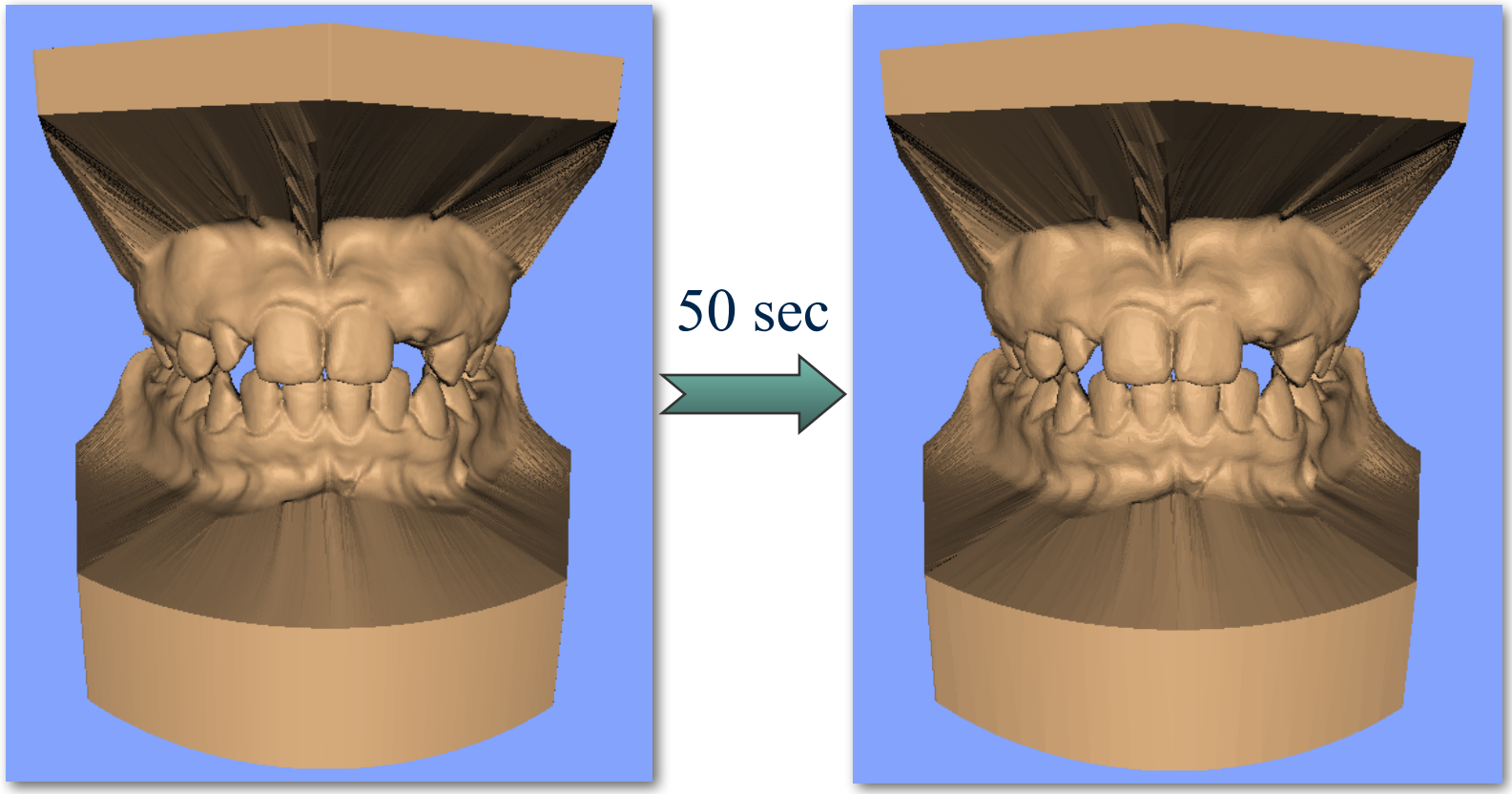
# Iterative Edge Contraction

Greedily apply edge contractions:

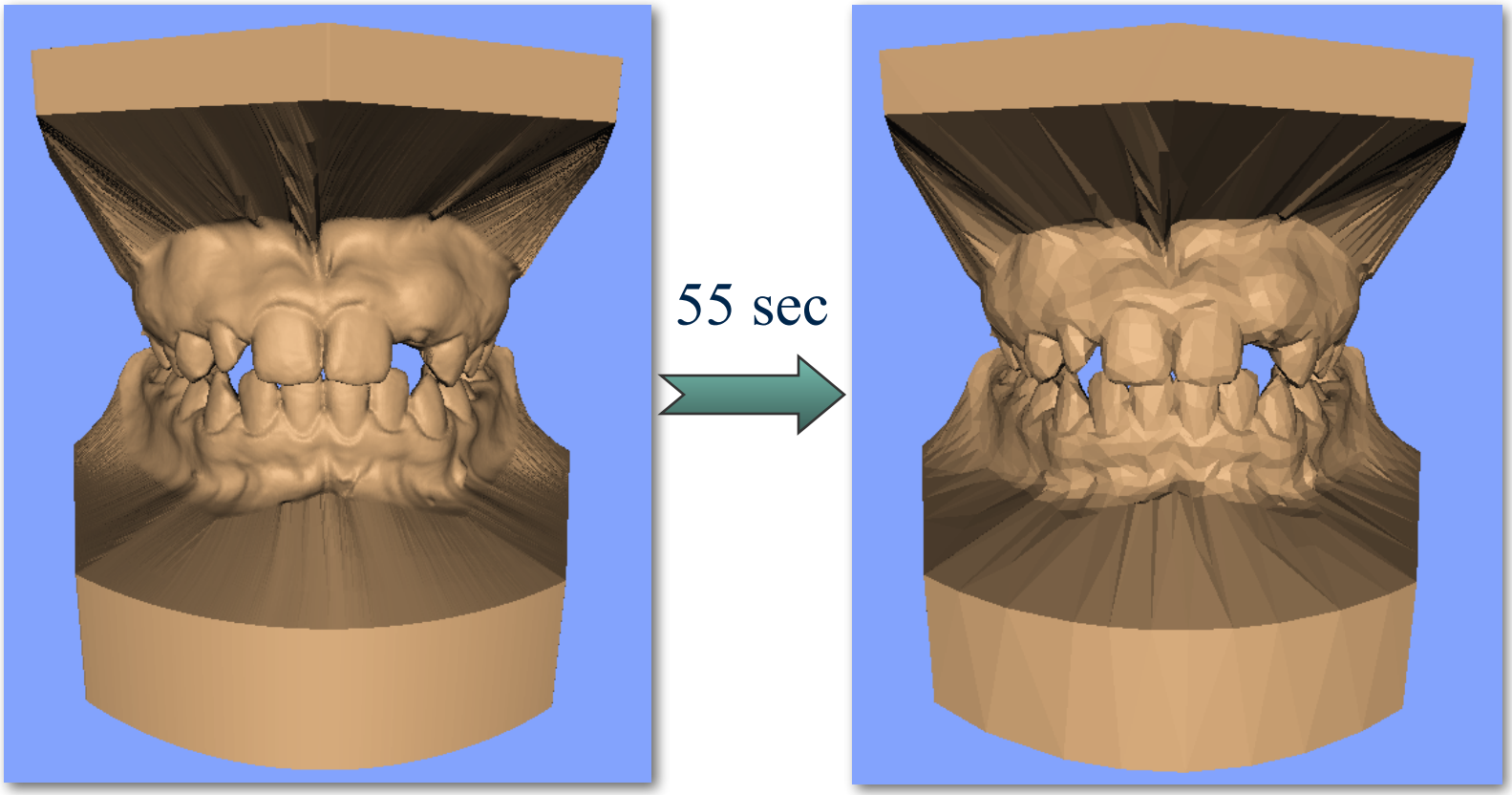
1. Rank all possible edge contractions with error it introduces
2. Contract edge with least error
3. Repeat until model is reduced to desired polygon count

Note: This does not produce optimal meshes (NP-hard problem)

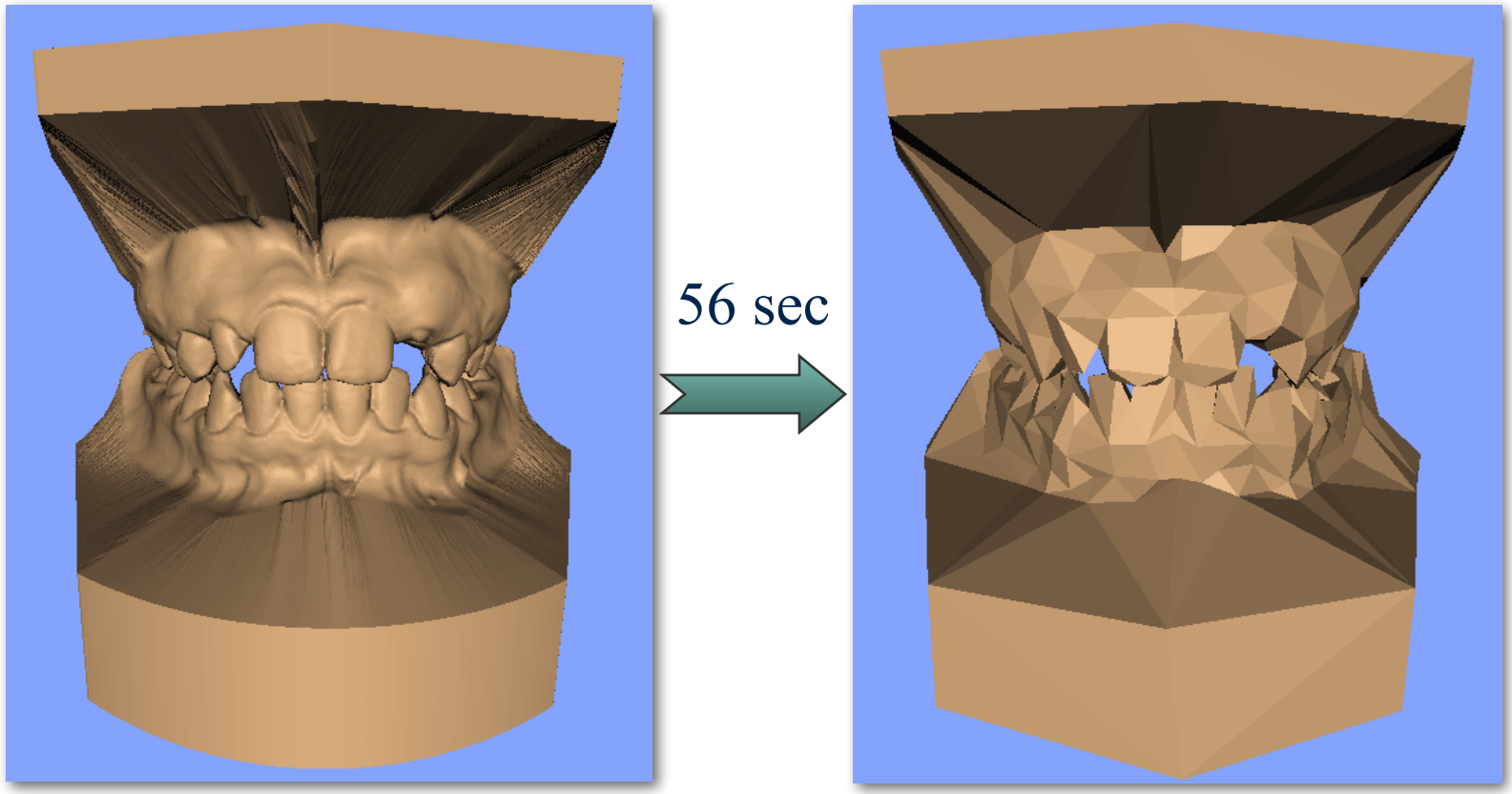
# LOD in Practice



# LOD in Practice



# LOD in Practice



# Subdivisions and Mesh Reduction

Good subdivision choices can allow for easy mesh reduction

- Artist works with low-poly base mesh and high-poly subdivided mesh simultaneously

In practice, though, specific tools still used for mesh reduction

- Greater control and more options
- Can consider other LOD issues (e.g. level streaming, LOD swapping, etc)

# Simplygon Demonstrations

Reduction:

https://www.youtube.com/watch?v=zTlJ58IMwG8

Remeshing:

https://www.youtube.com/watch?v=KieoxDq4Xak

# Tessellation Shader

Pipeline stage that allows for automatic subdivision on GPU

Involves three stages:

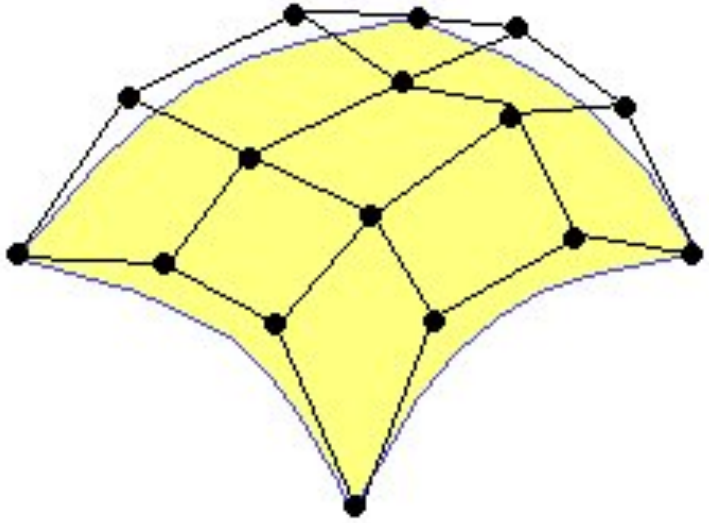
1. Hull or Tessellation Control Shader (TCS)
2. Tessellator or Primitive Generator (PG)
3. Domain Shader or Tessellation Evaluation Shader (TES)

Note: Naming differences due to DX versus OpenGL

# Tessellation Control Shader

Works on a group of vertices that define control points of surface geometry

Control points form patches



# Tessellation Control Shader

Takes patches as input

Emits output patches

Possible to apply transforms to patches and add or remove patches

# Tessellation Levels

Tessellation Levels determines number of triangles to generate per patch

- Allows for LOD based on camera distance, number of subdivisions etc

`gl_TessLevelInner` and `gl_TessLevelOuter` determine amount of tessellation per patch based on inner patches and outer edges

# Primitive Generator

Fixed function

Generates a domain of normalized subdivisions

- 2D square coordinates
- 3D barycentric coordinates

Note: Still does not have access to the actual patches

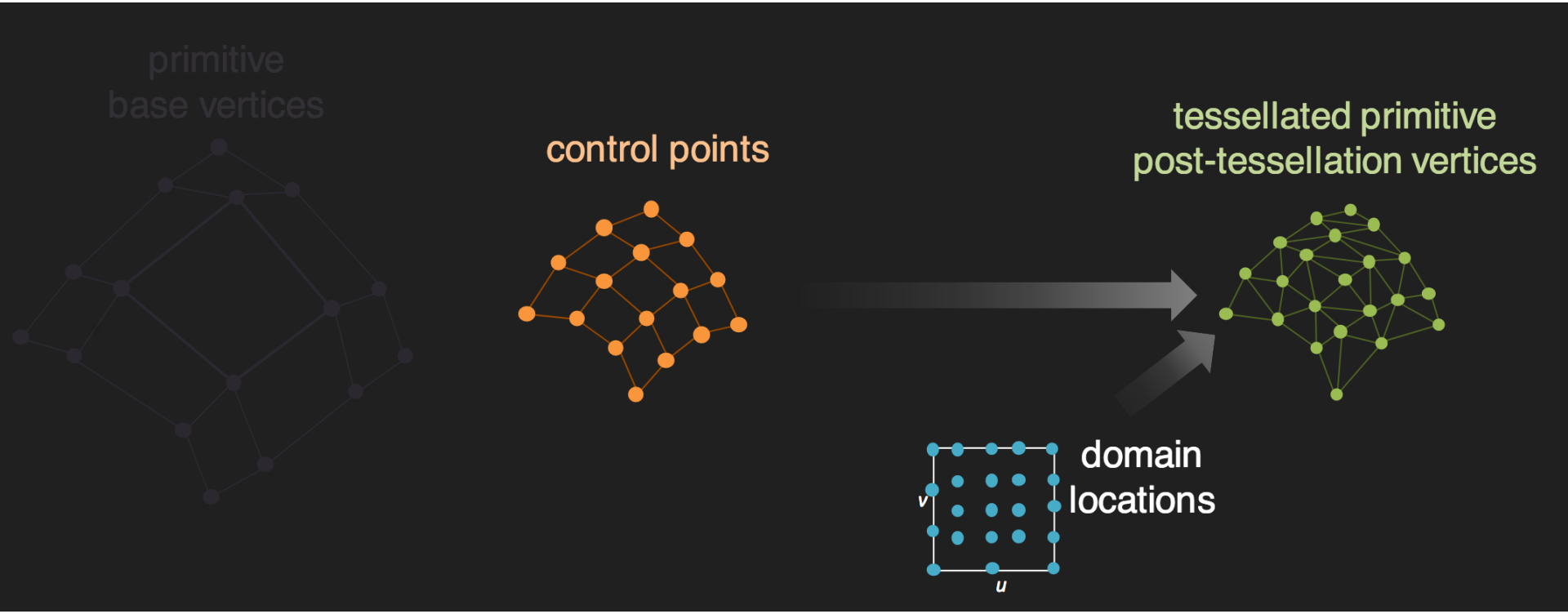
# Tessellation Evaluation Shader

Takes information from the TCS, the Domain generated in PG, and the patch information (position, normal, etc)

Creates one vertex for each barycentric coordinate based on TCS polynomial and patch

Tessellated vertices passed down the pipeline

# Pipeline Overview




https://developer.download.nvidia.com/assets/gameworks/downloads/regular/GDC17/RealTimeRenderingAdvances_Subdivision_GDC17_FINAL.pdf

# Tessellation Shader in Industry

- One of the newer pieces of the shader pipeline
- Allows for interactive subdivision functionality
- Integrated into Pixar's OpenSubdiv library:

http://graphics.pixar.com/opensubdiv/docs/intro.html

# LOD Switching

Popping is the sudden change in appearance as models swap

Flickering is the back-and-forth change between two resolutions at switching distance

# Reducing Popping

Create additional models at intermediate resolutions

Change distance of swap

Might also be an issue with texture streaming

# Reducing Flickering

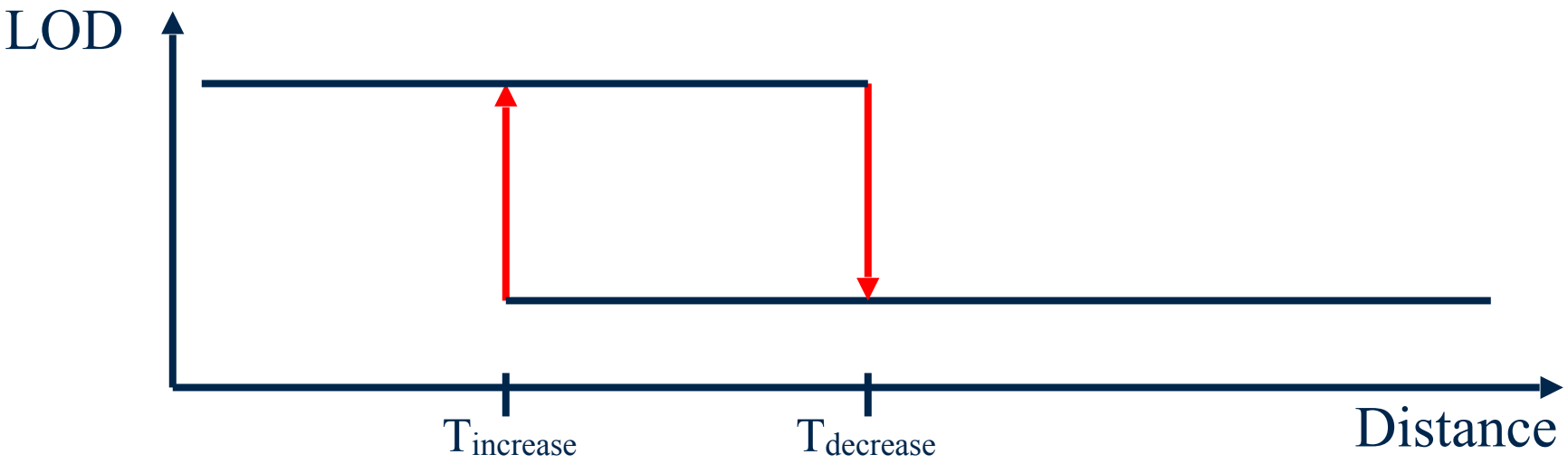
Show blended combination of model

- Image blending (alpha blending)
- Geometric blending (geomorphing)

Define two distinct thresholds for switching

- One determines distance for refinement, the other for reduction

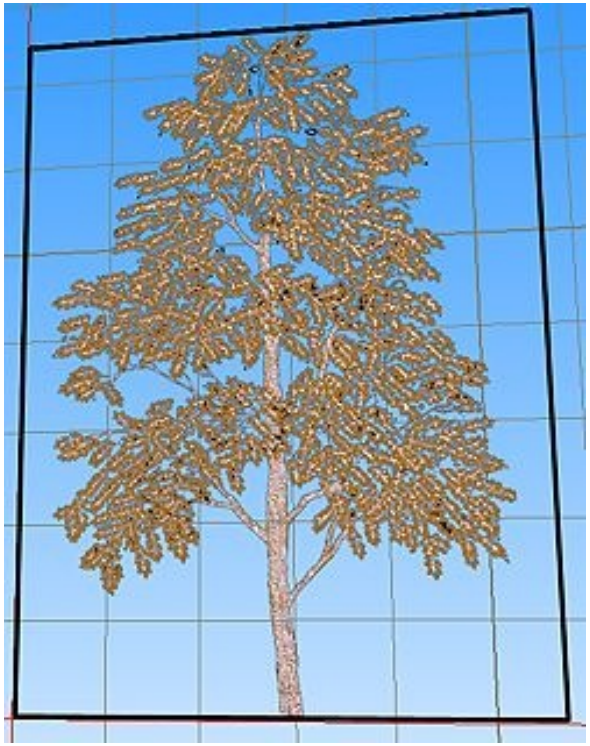
# Hysteresis Threshold



# Billboards

Idea: Extreme LOD to reduce all geometry to one or more textured polygons

- Considered image-based rendering
- Sometimes called sprites



# Generating Billboards

By hand – a skilled artist does the work!

- Paints color and alpha
- Generate a sequence of textures to animate

Automatically:

- Render a complex model and capture images
- Detect alpha by looking for background pixels
- Blend alpha at boundary for good anti-aliasing

# Billboard Configurations

Billboard polygons layouts:

- Single rectangle
- Two rectangles at right angles
- Several rectangles about a common axis
- Several rectangles stacked

# Single Polygon

Billboard consists of a single textured polygon

- What happens when it's not pointed at the viewer?

How can we solve this?

# Billboard Orientation

Point Sprites

- Billboard rotated about a central point that faces the camera

Axis Billboards

- Billboard aligned along an axis (arbitrary or axis-aligned)

# Aligning a Billboard

Billboard has a “forward” vector **F**

Billboard has an “up” vector **A**

Viewer has direction **V**

Goal: determine the angle to rotate the forward by to orient with the viewer

# Computing New Forward

Calculate **D**:

$$\boldsymbol{D} = \boldsymbol{A} \times \left( \boldsymbol{V} \times \boldsymbol{A} \right)$$

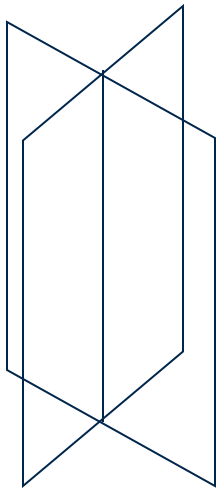
Compute angle $\gamma$ between **F** and **D**:

$$\gamma = \cos^{-1}\left( \frac{\boldsymbol{F} \bullet \boldsymbol{D}}{\|\boldsymbol{F}\| \|\boldsymbol{D}\|} \right)$$

# Multi-Polygon Billboards

Use two polygons at right angles

- No alignment with viewer
- What is this good for?

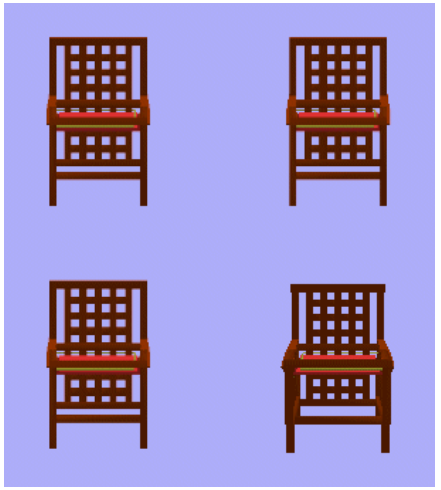


More polygons look better

Can render by blending or using depth buffer

# View Dependent Billboards

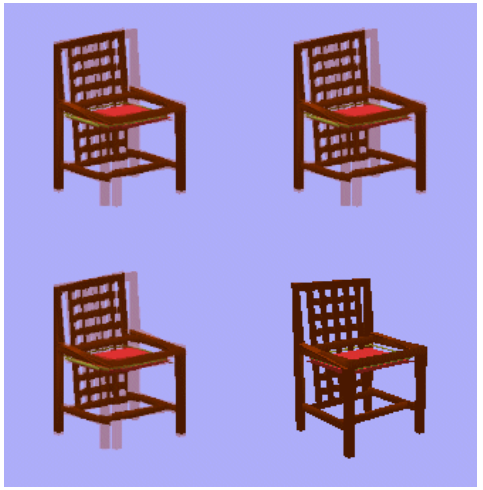
For objects that are not rotationally symmetric



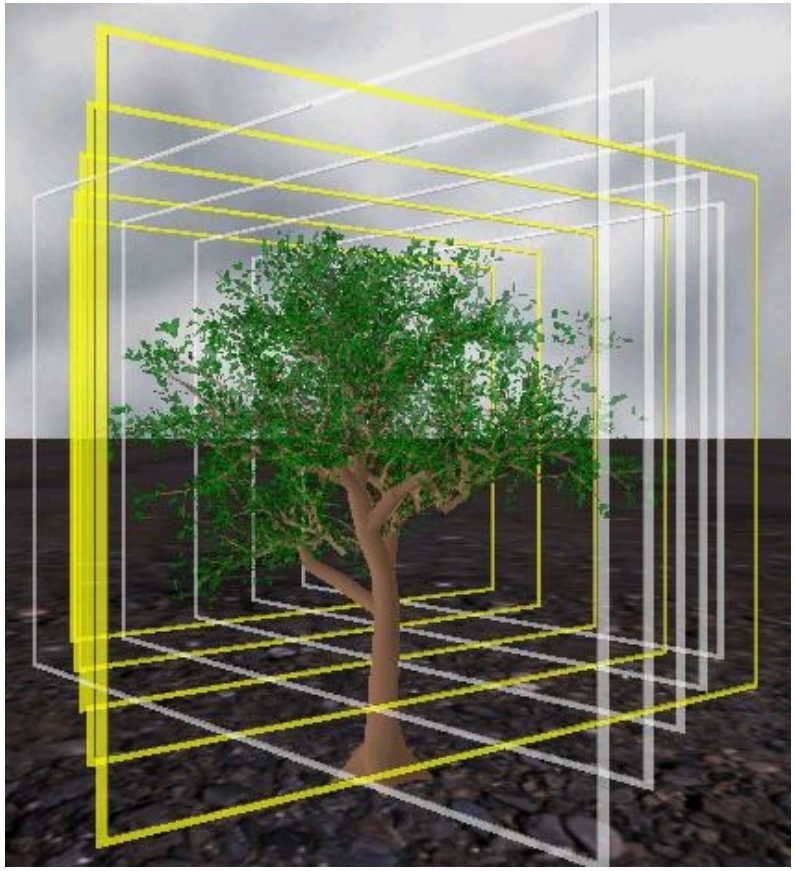
Compute multiple textures for multiple view points

Fix polygon but vary texture

Can use 3D textures and hardware texture filtering



# Imposter Example



# Reducing Geometry

Ways to reduce geometry:

1. Reduce the number of triangles
   - Visibility culling
   - Level of detail
2. Reduce amount of data sent per triangle
   - Mesh compression

# Why Compress Meshes?

Primarily for networked applications (helps with memory bandwidth)

Observation: Vertex data includes position, color, texture, normal, etc

Much of this data is redundant

- Triangles share vertices
- Vertices share colors, etc

# Mesh Compression

Pipeline hardware usually has small buffers

- Accepts data in a stream

Must decompress in software

- Handle triangle connectivity separately from vertex attributes
- Create long strips or implicit connectivity structures