Triangle meshes

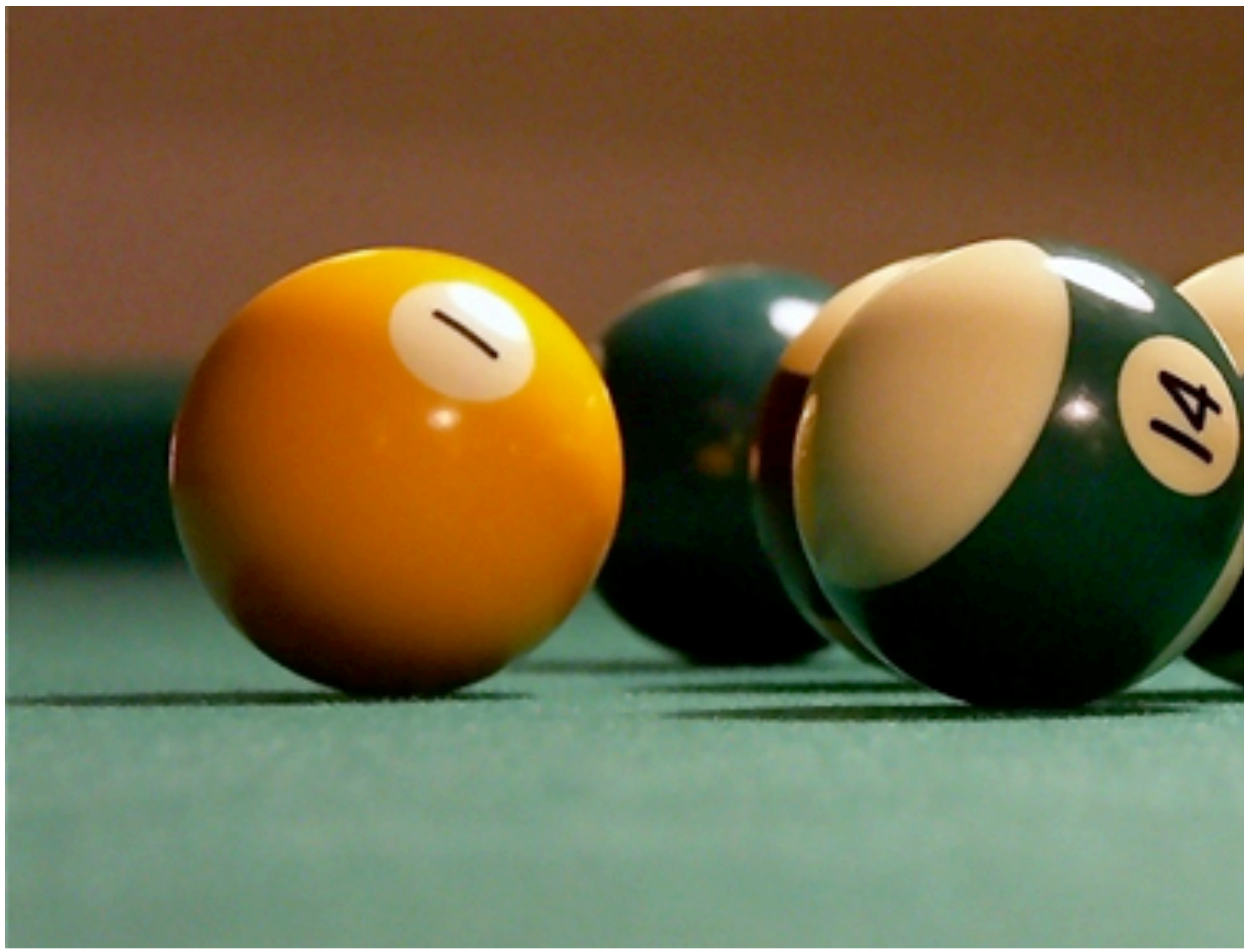
Steve Marschner

CS 4620

Cornell University

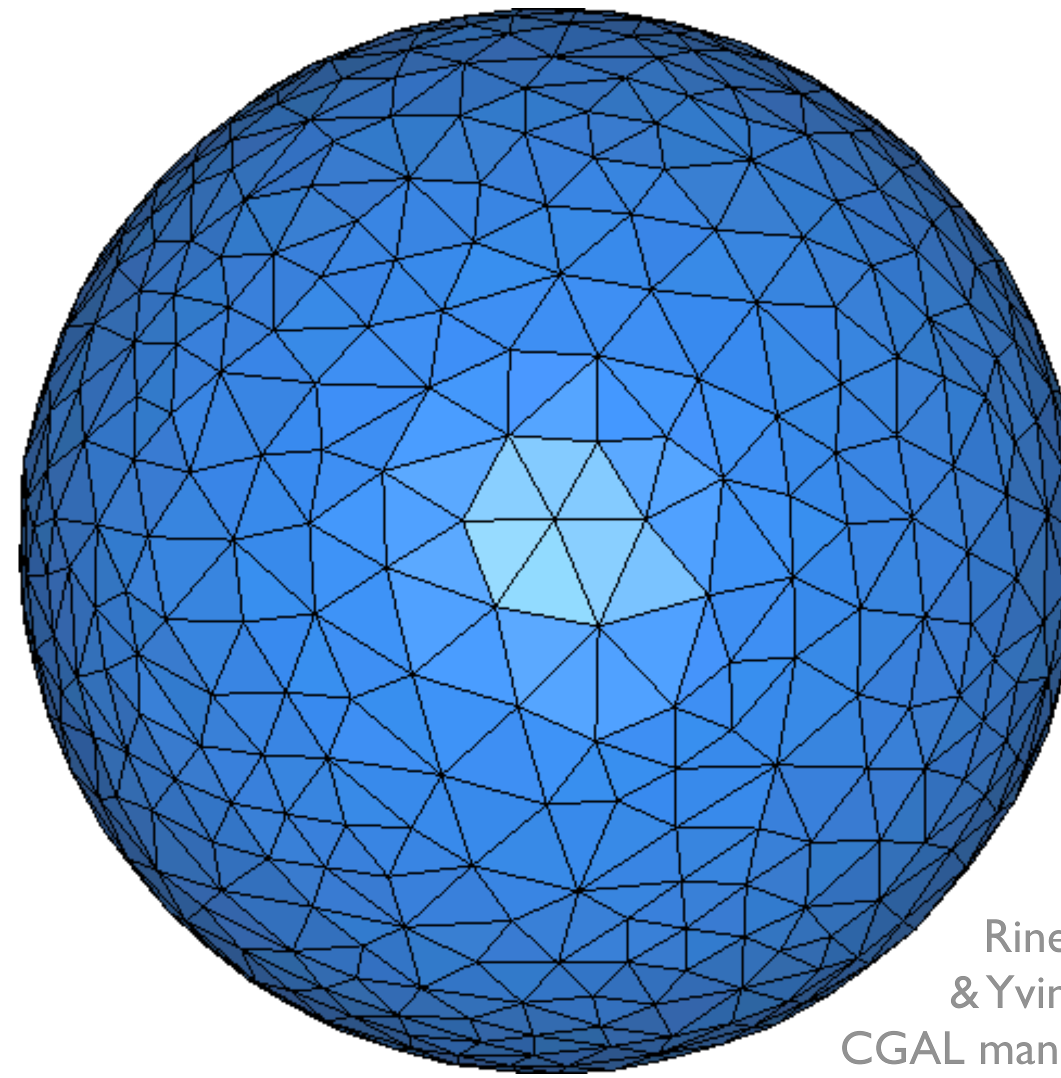
Triangle meshes

1. Meshes and mesh representation



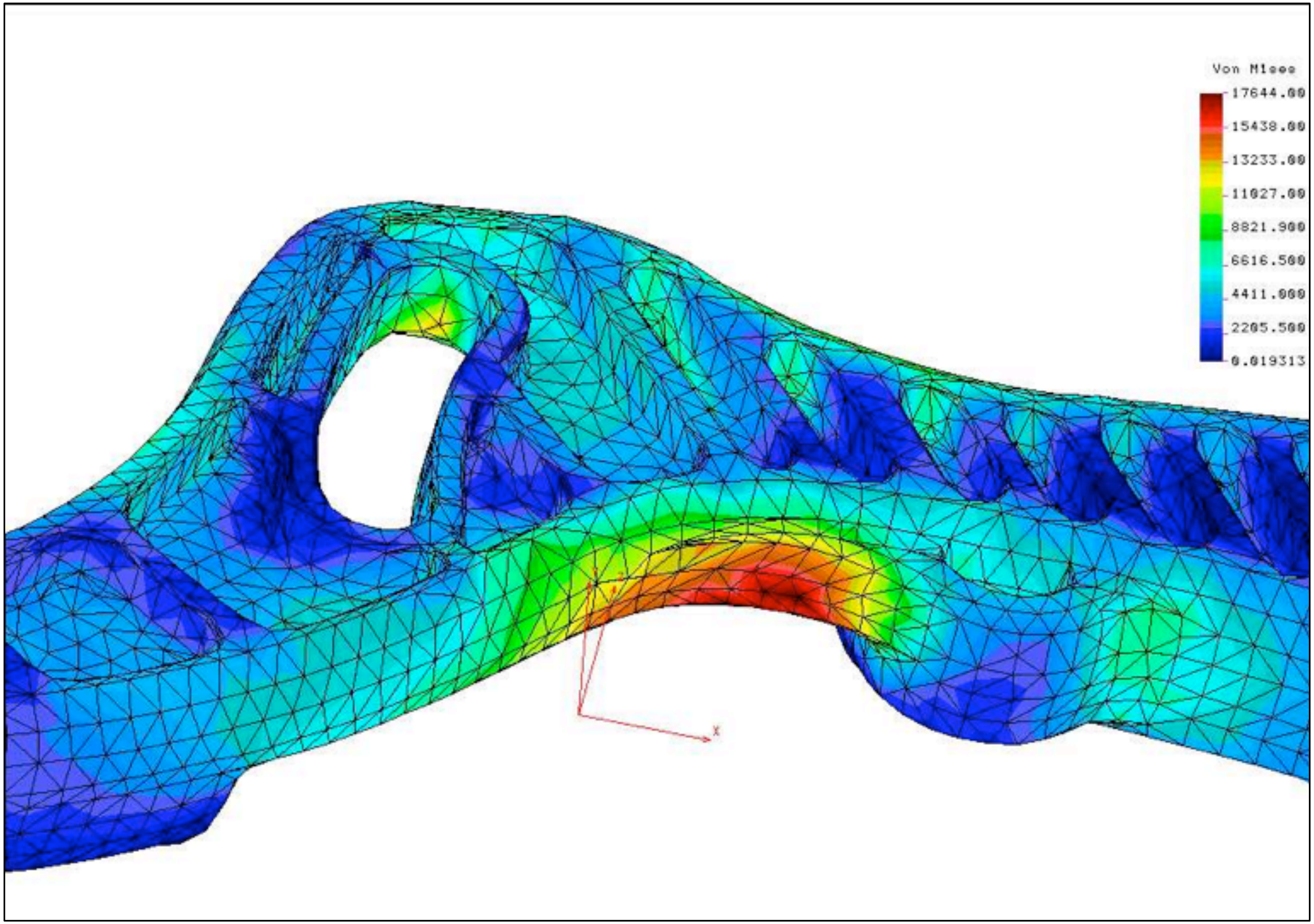
Andrzej Barabasz

spheres



Rineau & Yvinec CGAL manual

approximate sphere

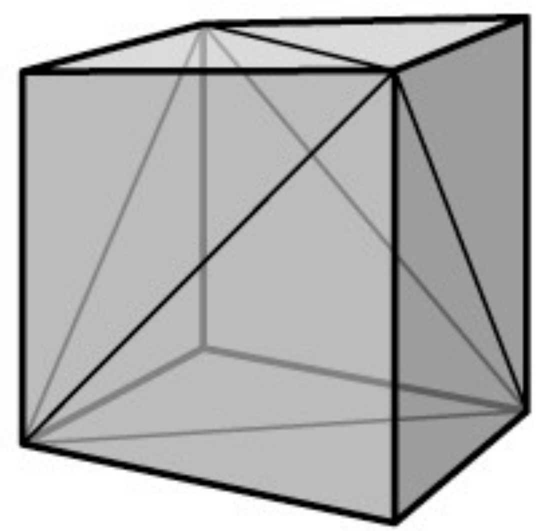


PATRIOT Engineering

finite element analysis

Ottawa Convention Center

A small triangle mesh

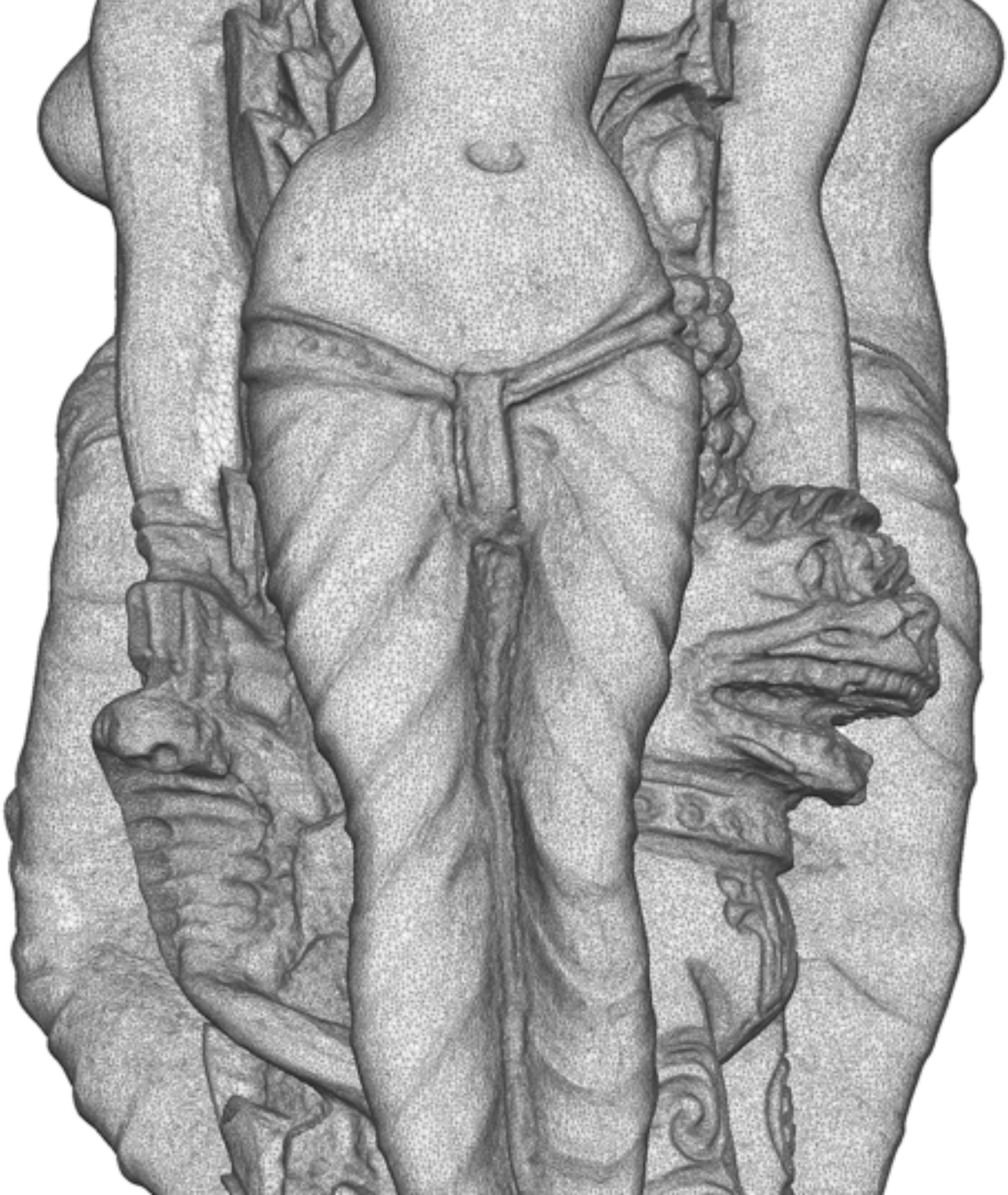


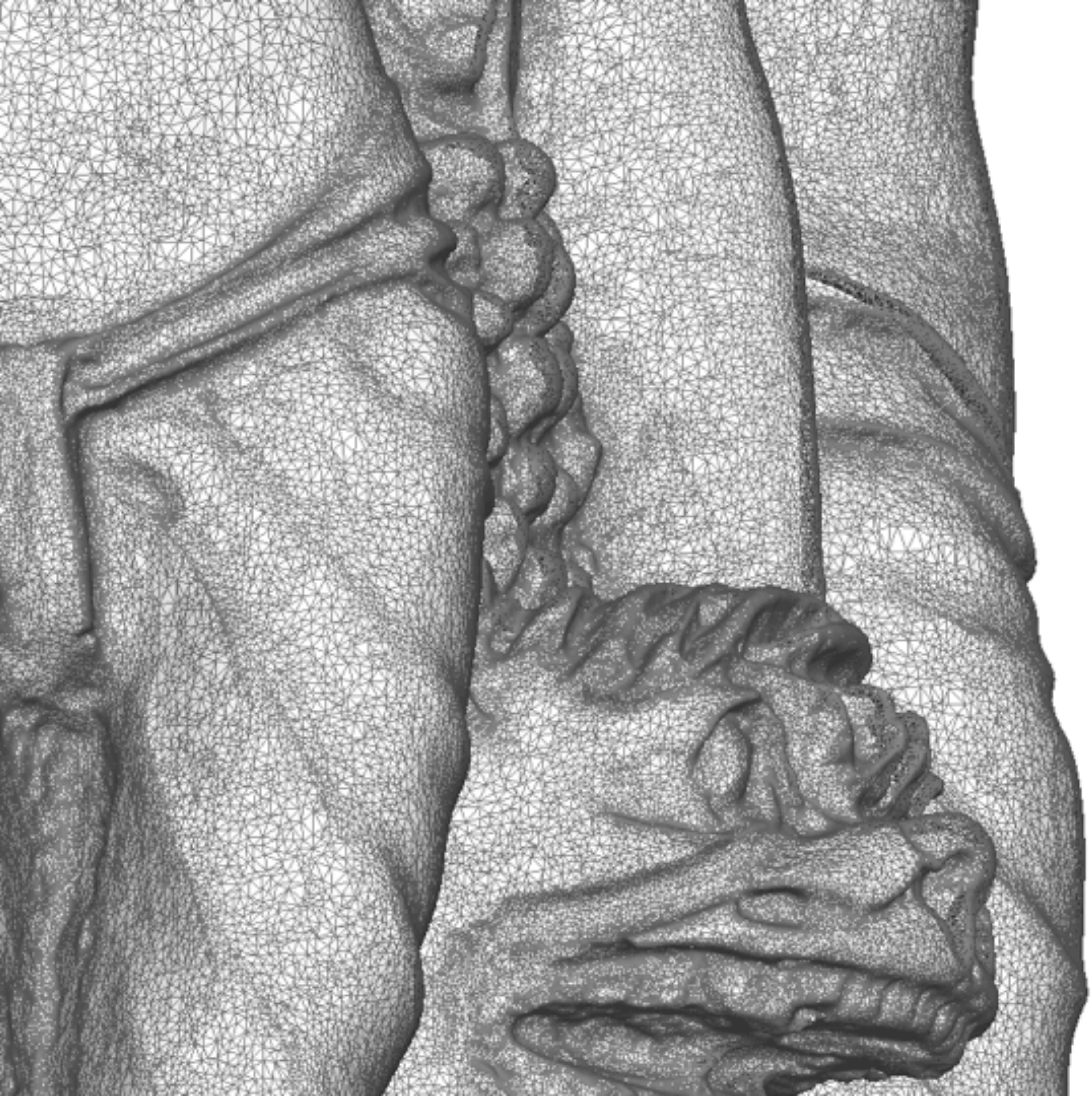
12 triangles, 8 vertices

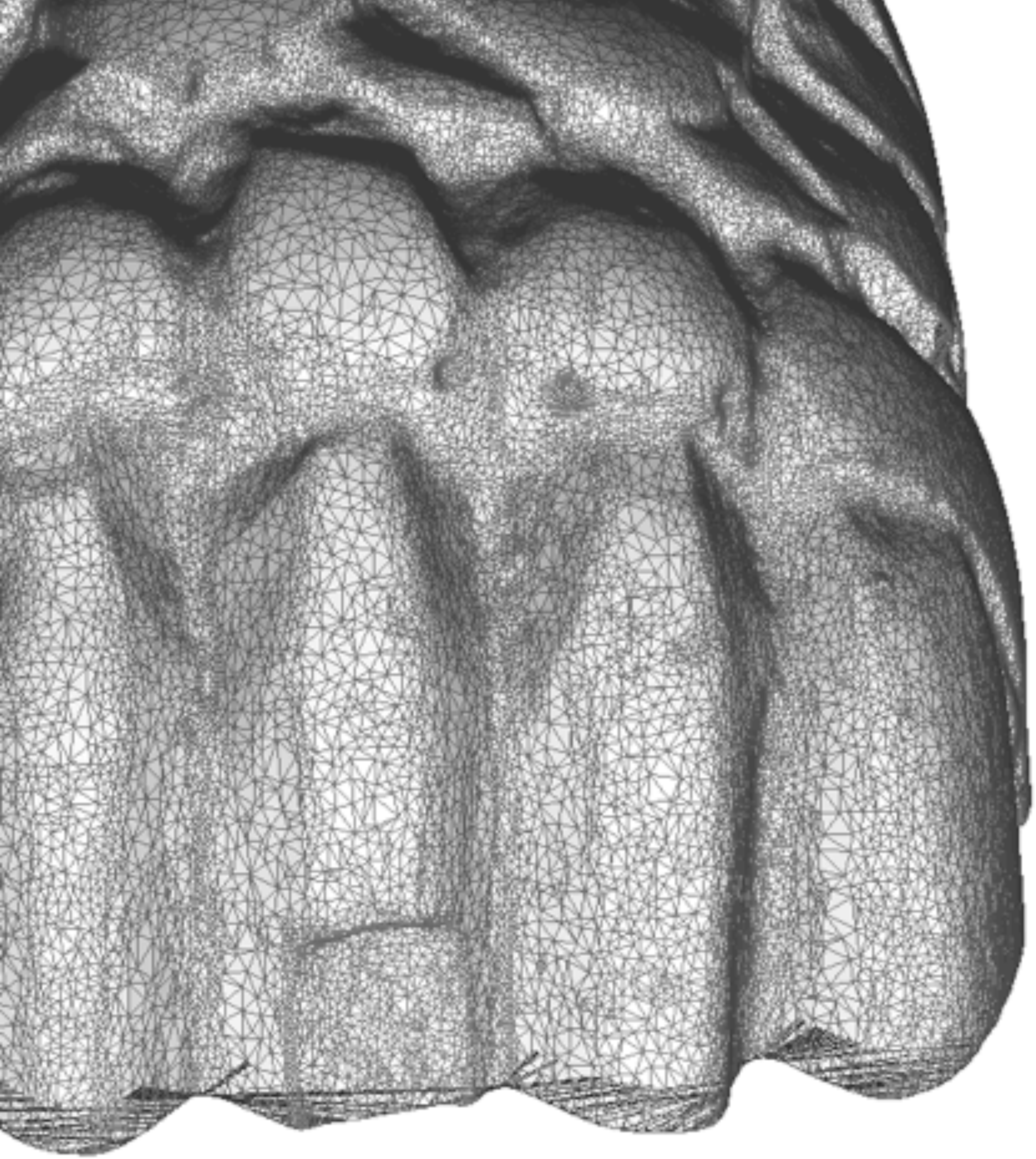
A large mesh

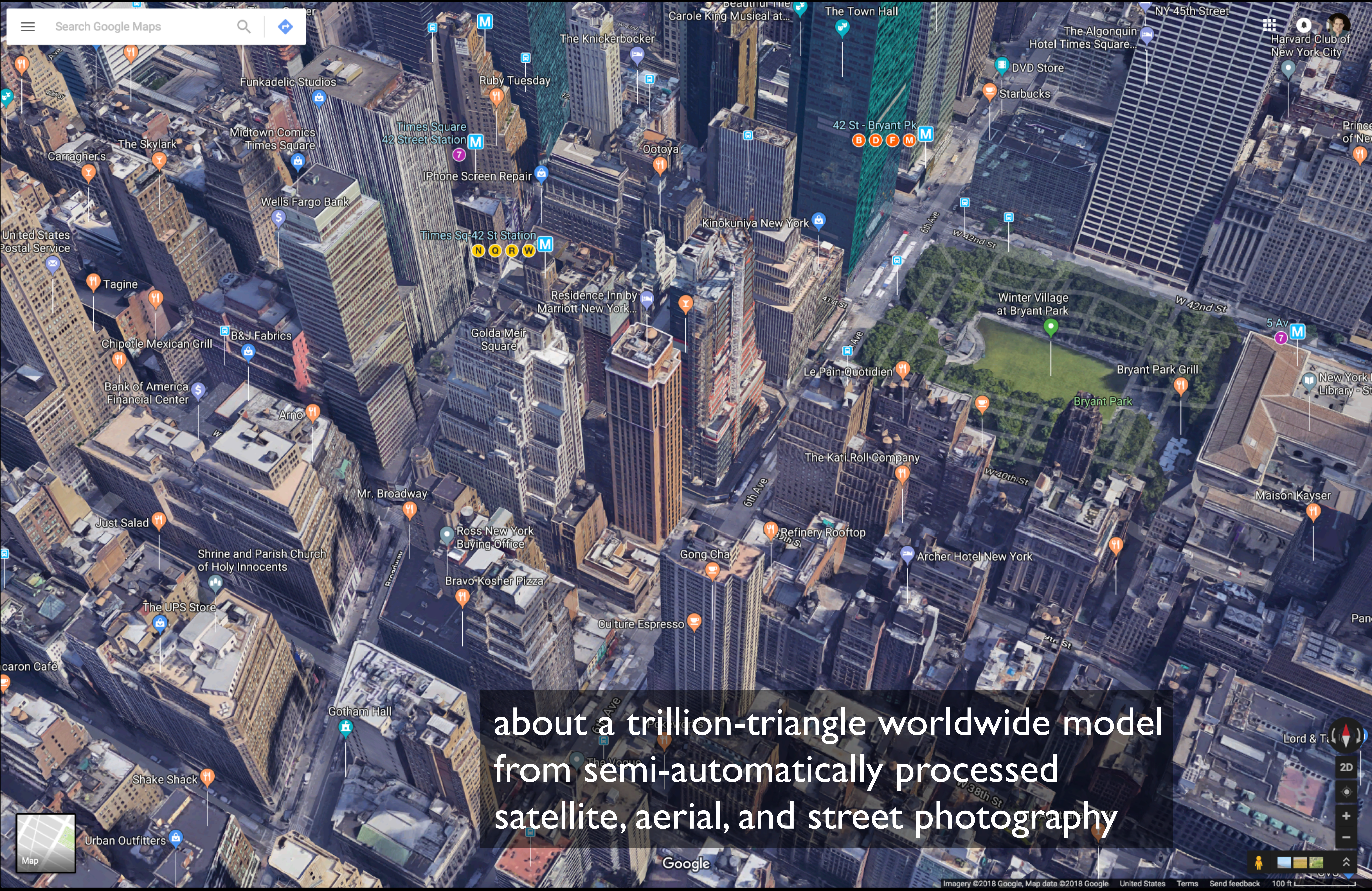
10 million triangles
from a high-resolution
3D scan

Traditional Thai sculpture—scan by XYZRGB, inc., image by MeshLab project









about a trillion-triangle worldwide model from semi-automatically processed satellite, aerial, and street photography

Triangles

- **Defined by three *vertices***
- **Lives in the plane containing those vertices**
- **Vector normal to plane is the triangle's normal**
- **Conventions (for this class, not everyone agrees):**
  – vertices are counter-clockwise as seen from the "outside" or "front"
  – surface normal points towards the outside ("outward facing normals")

Triangle meshes

- **A bunch of triangles in 3D space that are connected together to form a surface**
- **Geometrically, a mesh is a *piecewise planar* surface**
  - almost everywhere, it is planar
  - exceptions are at the edges where triangles join
- **Often, it's a piecewise planar approximation of a smooth surface**
  - in this case the creases between triangles are artifacts—we don't want to see them

Representation of triangle meshes

- **Compactness**
- **Efficiency for rendering**
  - enumerate all triangles as triples of 3D points
- **Efficiency of queries**
  - all vertices of a triangle
  - all triangles around a vertex
  - neighboring triangles of a triangle
  - (need depends on application)
    - finding triangle strips
    - computing subdivision surfaces
    - mesh editing

Representations for triangle meshes

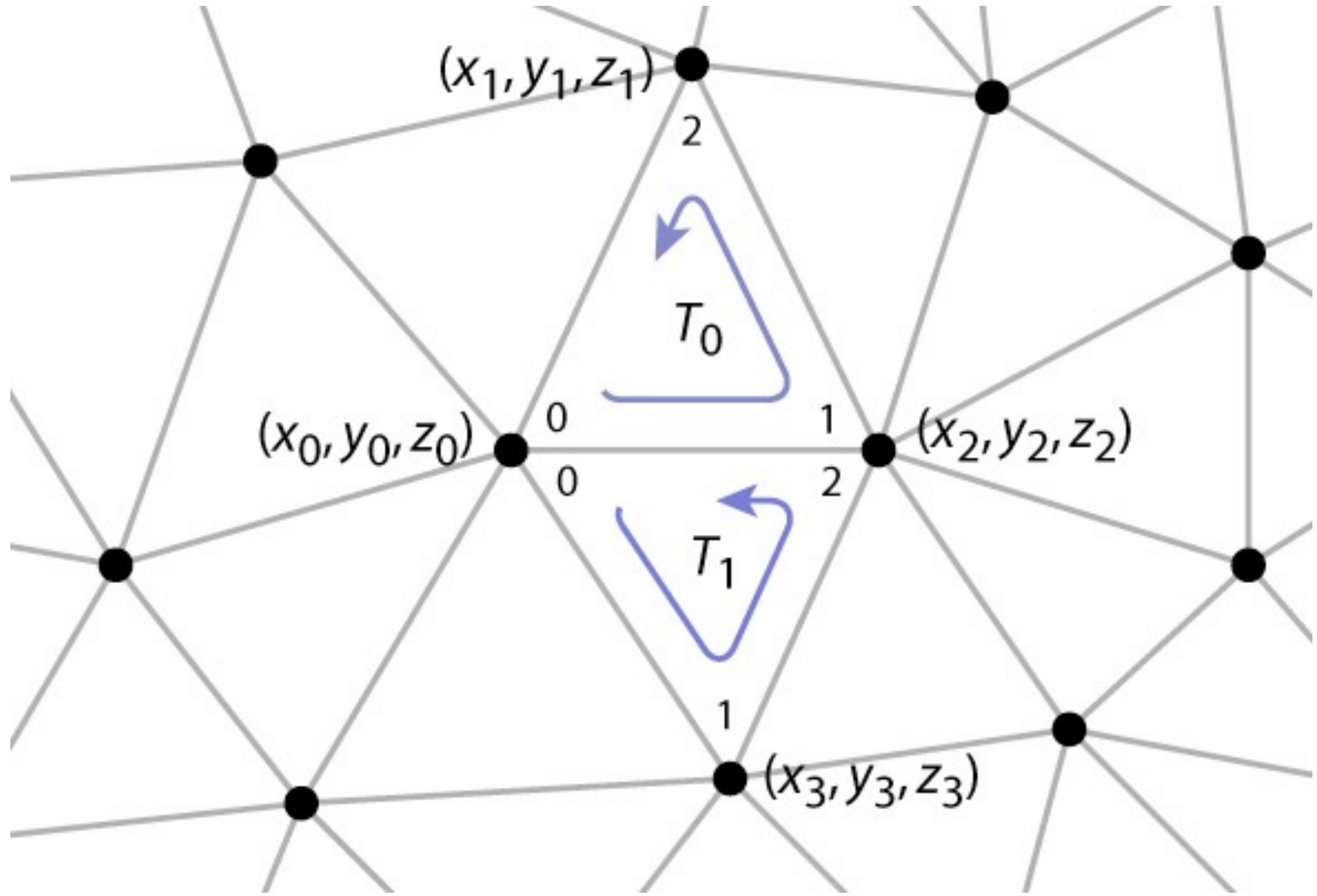
- **Separate triangles**
- **Indexed triangle set**
  - shared vertices
- **Triangle strips and triangle fans**
  - compression schemes for fast transmission
- **Triangle-neighbor data structure**
  - supports adjacency queries
- **Winged-edge data structure**
  - supports general polygon meshes

crucial for
rendering 2 assignment

Interesting and
useful but not
needed for
assignments

Separate triangles

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮



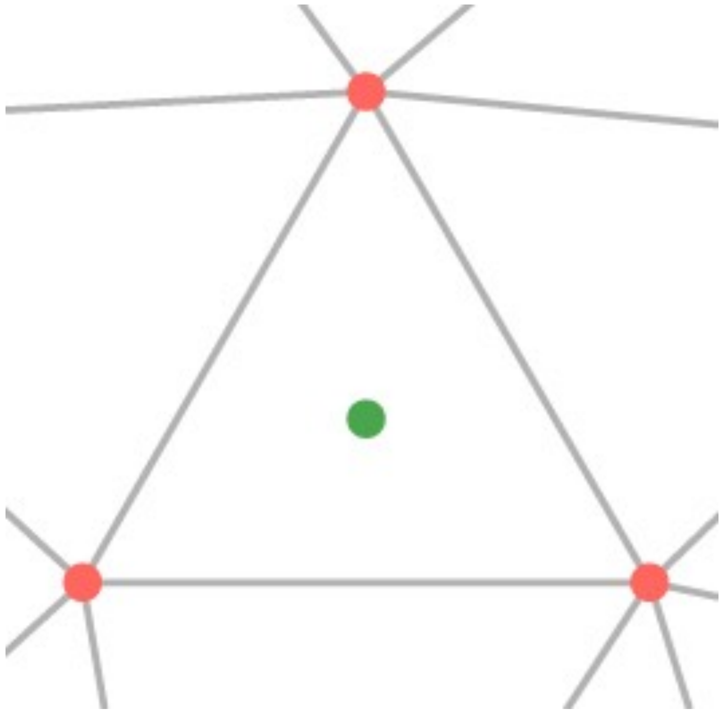
Separate triangles

- **array of triples of points**
  - float[n_T][3][3]: 36 bytes per triangle
    - 3 vertices per triangle
    - 3 coordinates per vertex
    - 4 bytes per coordinate (float)
- **various problems**
  - wastes space (each vertex stored 6 times)
  - cracks due to roundoff
  - difficulty of finding neighbors at all

Indexed triangle set

- **Store each vertex once**
- **Each triangle points to its three vertices**

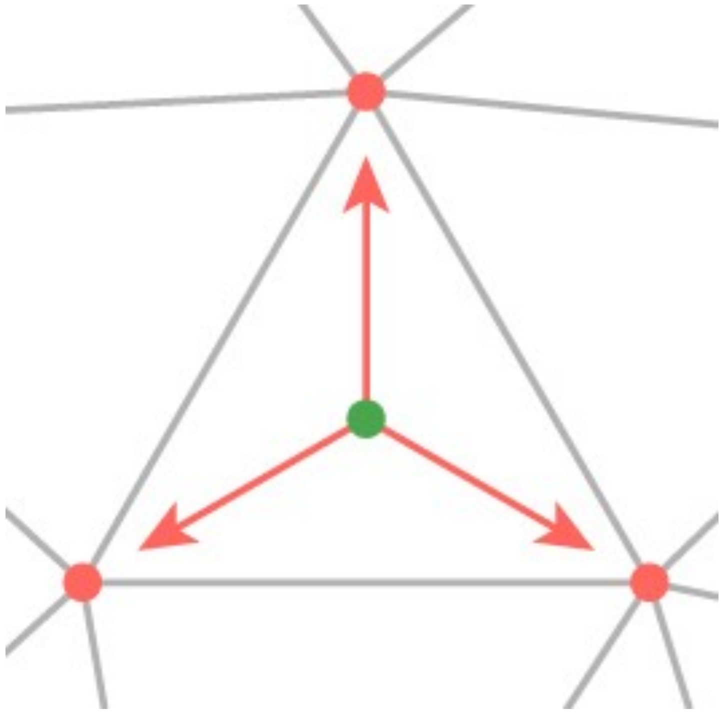
```
Triangle {
  Vertex vertex[3];
  }

Vertex {
  float position[3];  // or other data
  }

// ... or ...

Mesh {
  float verts[nv][3];  // vertex positions (or other data)
  int tInd[nt][3];  // vertex indices
  }
```



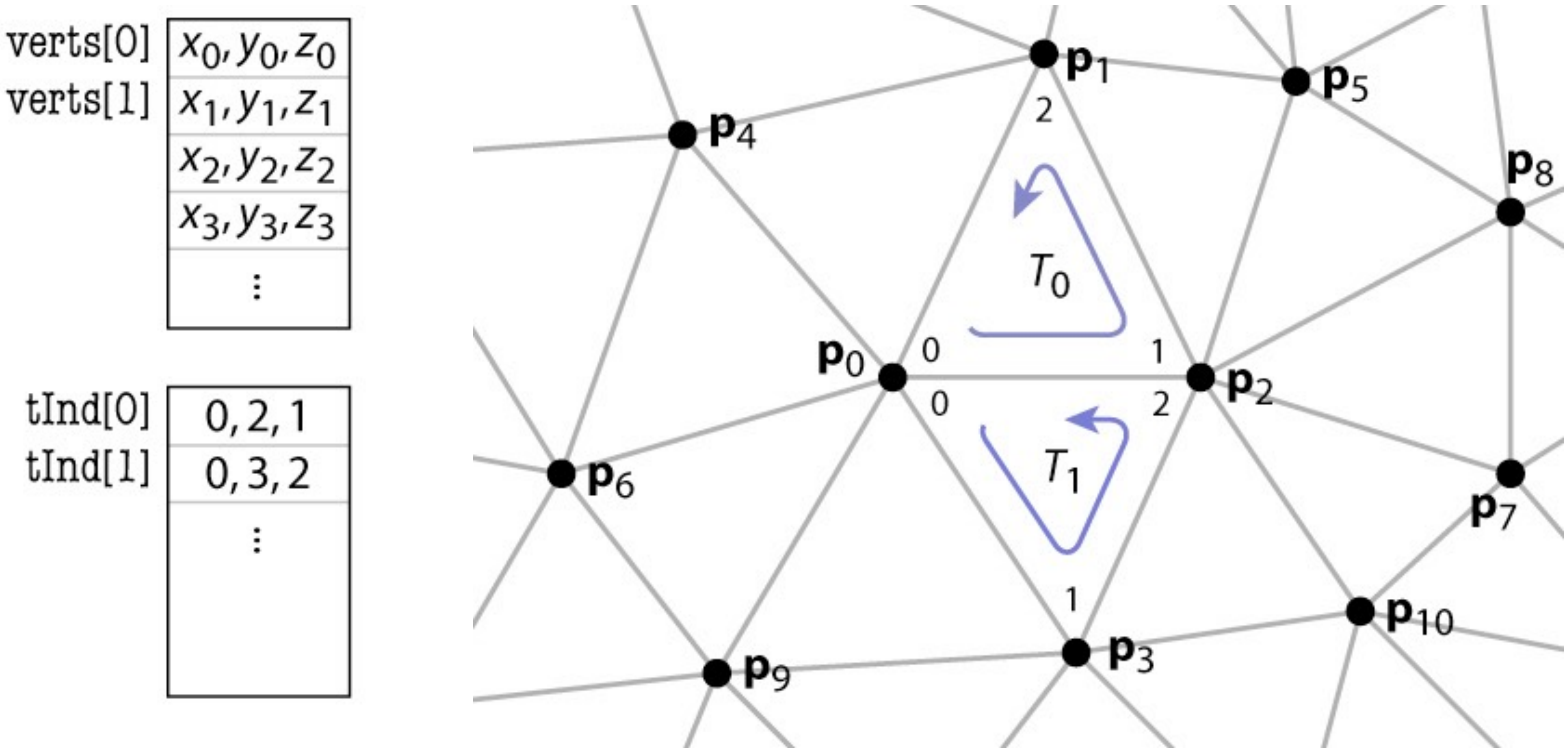
Indexed triangle set

- **Store each vertex once**
- **Each triangle points to its three vertices**

```
Triangle {
  Vertex vertex[3];
  }

Vertex {
  float position[3];  // or other data
  }

// ... or ...

Mesh {
  float verts[nv][3];  // vertex positions (or other data)
  int tInd[nt][3];  // vertex indices
  }
```

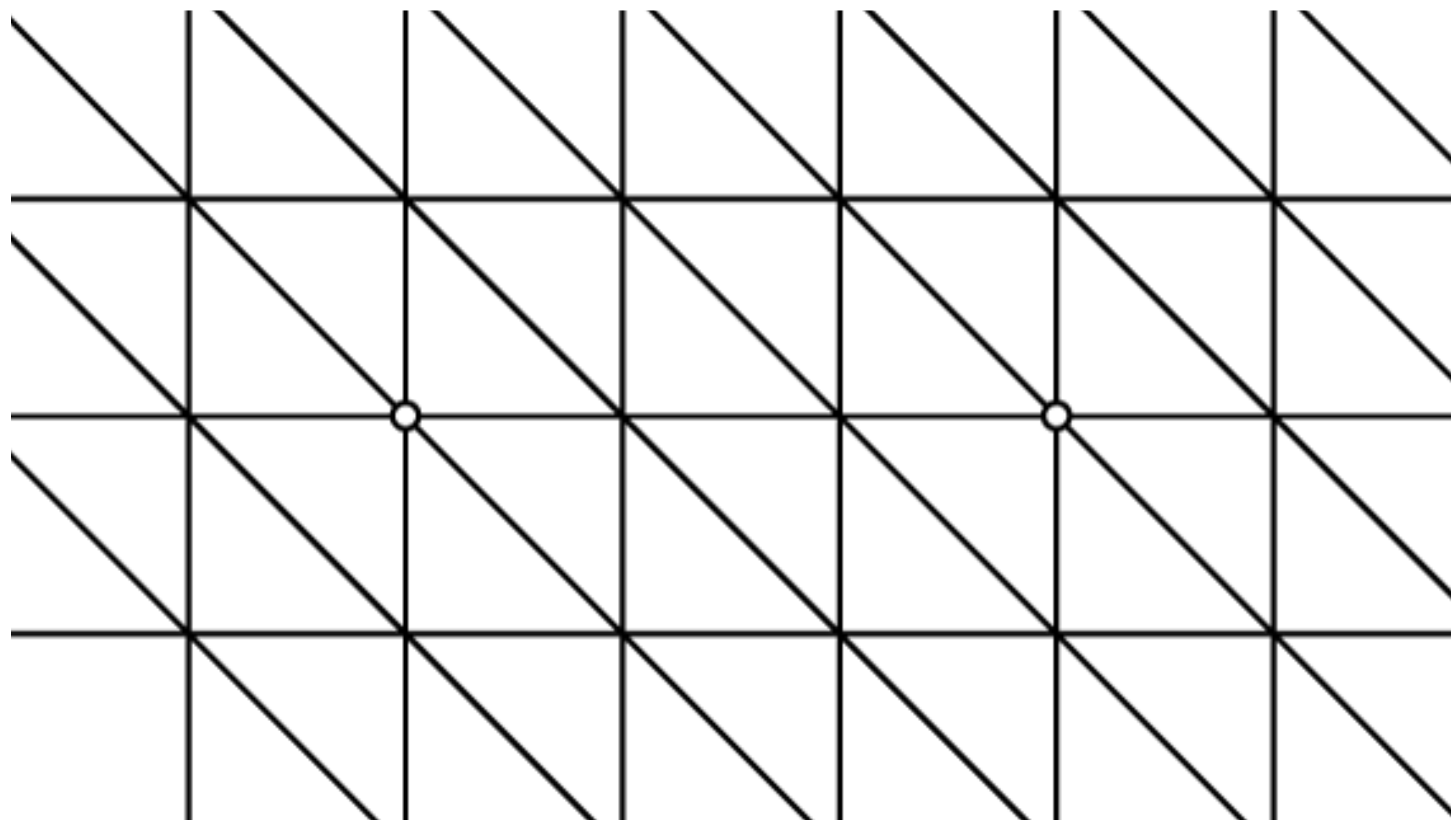


Indexed triangle set



Estimating storage space

- n_T = #tris; n_V = #verts; n_E = #edges
- Rule of thumb: n_T:n_E:n_V is about 2:3:1



[Alec Jacobson]

Indexed triangle set

- **array of vertex positions**
  - float[n_V][3]: 12 bytes per vertex
    - (3 coordinates x 4 bytes) per vertex
- **array of triples of indices (per triangle)**
  - int[n_T][3]: about 24 bytes per vertex
    - 2 triangles per vertex (on average)
    - (3 indices x 4 bytes) per triangle
- **total storage: 36 bytes per vertex (factor of 2 savings)**
  - separate triangles: 36 bytes per *triangle*
- **represents topology and geometry separately**
- **finding neighbors is at least well defined**

Practical encoding of meshes

- **OBJ file format**
  - widely used format for polygon meshes in indexed form
  - supports the usual attributes: position, normal, texture coordinate
  - allows triangles or polygons (only triangles and quads widely supported)
  - comes with a crude mechanism for adding materials
- **Demo**
  - simple file with one triangle

Triangle meshes

1. Storing data at vertices

Data on meshes

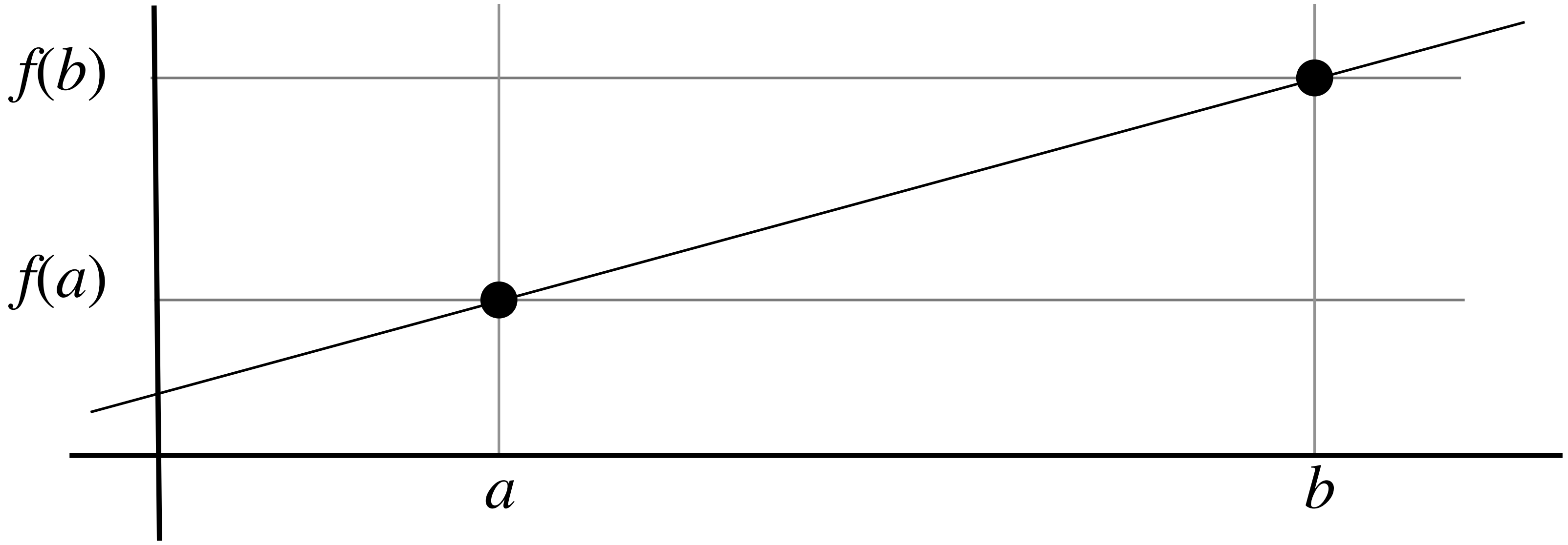
- **Often need to store additional information besides just the geometry**
- **Can store additional data at faces, vertices, or edges**
- **Examples**
  - colors stored on faces, for faceted objects
  - information about sharp creases stored at edges
  - any quantity that varies *continuously* (without sudden changes, or *discontinuities*) gets stored at vertices

Key types of vertex data

- **Surface normals**
  - when a mesh is approximating a curved surface, store normals at vertices
- **Surface parameterizations aka. texture coordinates**
  - providing a 2D coordinate system on the surface
- **Positions**
  - at some level this is just another piece of data
  - position varies continuously between vertices
- **Vertex data defines a continuous function on the mesh surface**
  - quantities are defined in triangle interiors via linear interpolation

Linear interpolation, 1D domain

- **Given values of a function f(x) for two values of x, you can define in-between values by drawing a line**



See textbook Sec. 2.6

- there is a unique line through the two points
- can write down using slopes, intercepts
- …or as a value added to f(a)
- …or as a convex combination of f(a) and f(b)

$$\begin{aligned} f(x) &= f(a) + \frac{x-a}{b-a}(f(b) - f(a)) \\ &= (1-\beta)f(a) + \beta f(b) \\ &= \alpha f(a) + \beta f(b) \end{aligned}$$

Linear interpolation in 1D

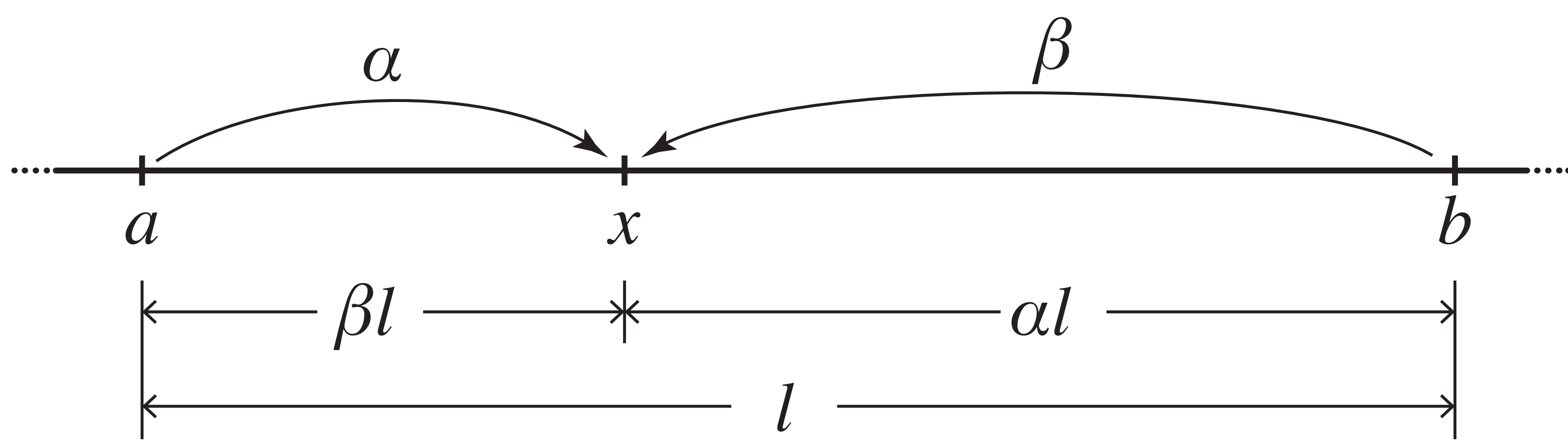
- **Alternate story**
  1. write *x* as convex combination of a and b

     $$x = \alpha a + \beta b \quad \text{where } \alpha + \beta = 1$$

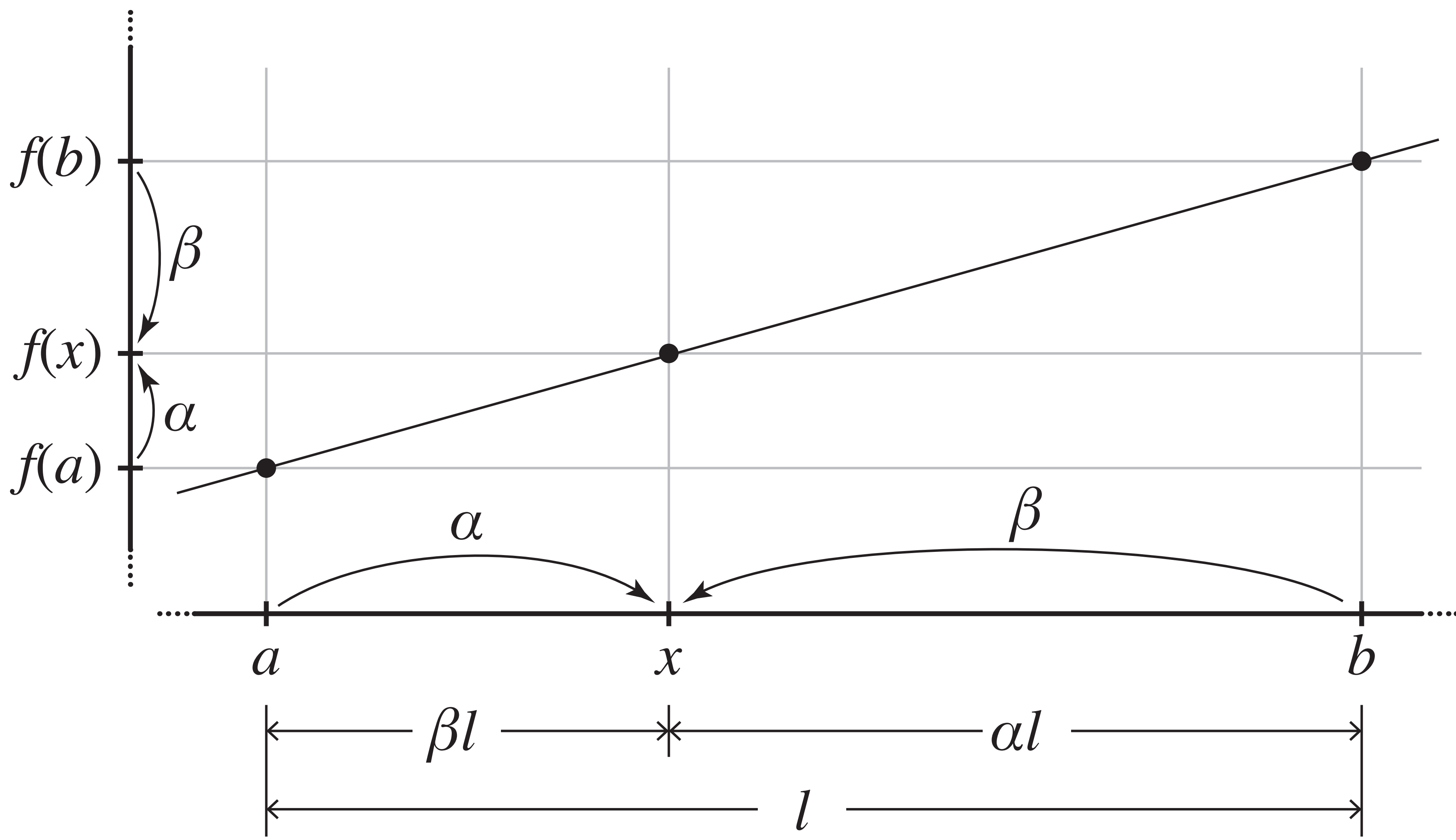
  2. use the same weights to compute ***f***(***x***) as a convex combination of ***f***(***a***) and ***f***(***b***)

     $$f(x) = \alpha f(a) + \beta f(b)$$

Linear interpolation in 1D



Linear interpolation in 1D



Linear interpolation in 2D

- **Use the alternate story:**
  1. Write **x**, the point where you want a value, as a convex linear combination of the vertices

$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \quad \text{where } \alpha + \beta + \gamma = 1$$

  2. Use the same weights to compute the interpolated value *f*(**x**) from the values at the vertices, *f*(**a**), *f*(**b**), and *f*(**c**)

$$f(\mathbf{x}) = \alpha f(\mathbf{a}) + \beta f(\mathbf{b}) + \gamma f(\mathbf{c})$$

See textbook
Sec. 2.7

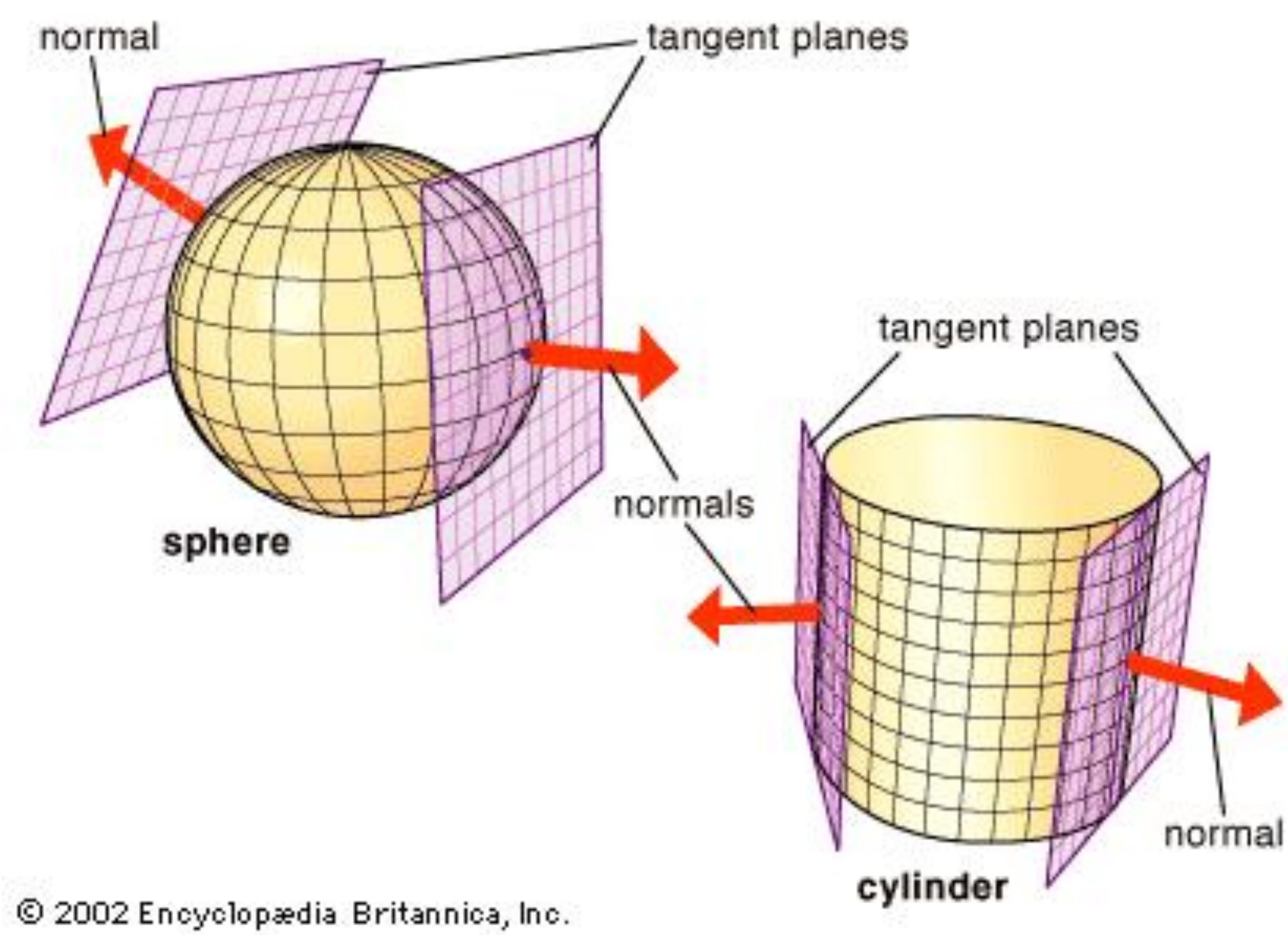
Interpolation in ray tracing

- **When values are stored at vertices, use linear (barycentric) interpolation to define values across the whole surface that:**
  1. …match the values at the vertices
  2. …are continuous across edges
  3. …are piecewise linear (linear over each triangle)

     as a function of 3D position, not screen position—more later
- **How to compute interpolated values**
  4. during triangle intersection compute barycentric coords
  5. use barycentric coords to average attributes given at vertices

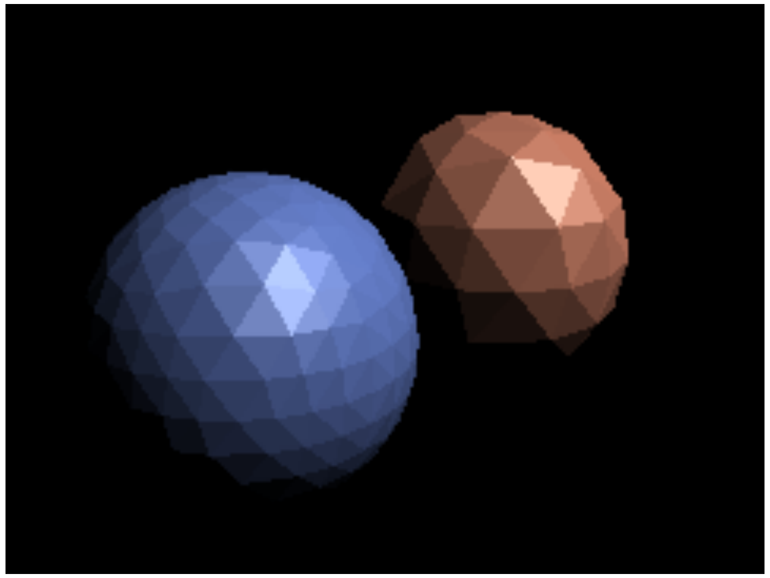
Differential geometry 101

- **Tangent plane**
  - at a point on a smooth surface in 3D, there is a unique plane tangent to the surface, called the *tangent plane*
- **Normal vector**
  - vector perpendicular to a surface (that is, to the tangent plane)
  - only unique for smooth surfaces (not at corners, edges)

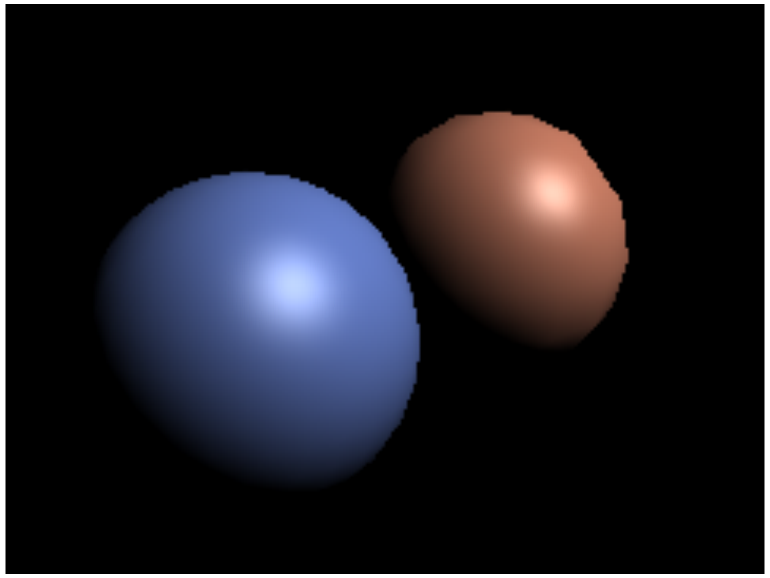


Surface normals on meshes

- **For smooth surfaces approximated with meshes**
- **Use interpolated normal for shading in place of actual normal**
  - "shading normal" vs. "geometric normal"



geometric normals



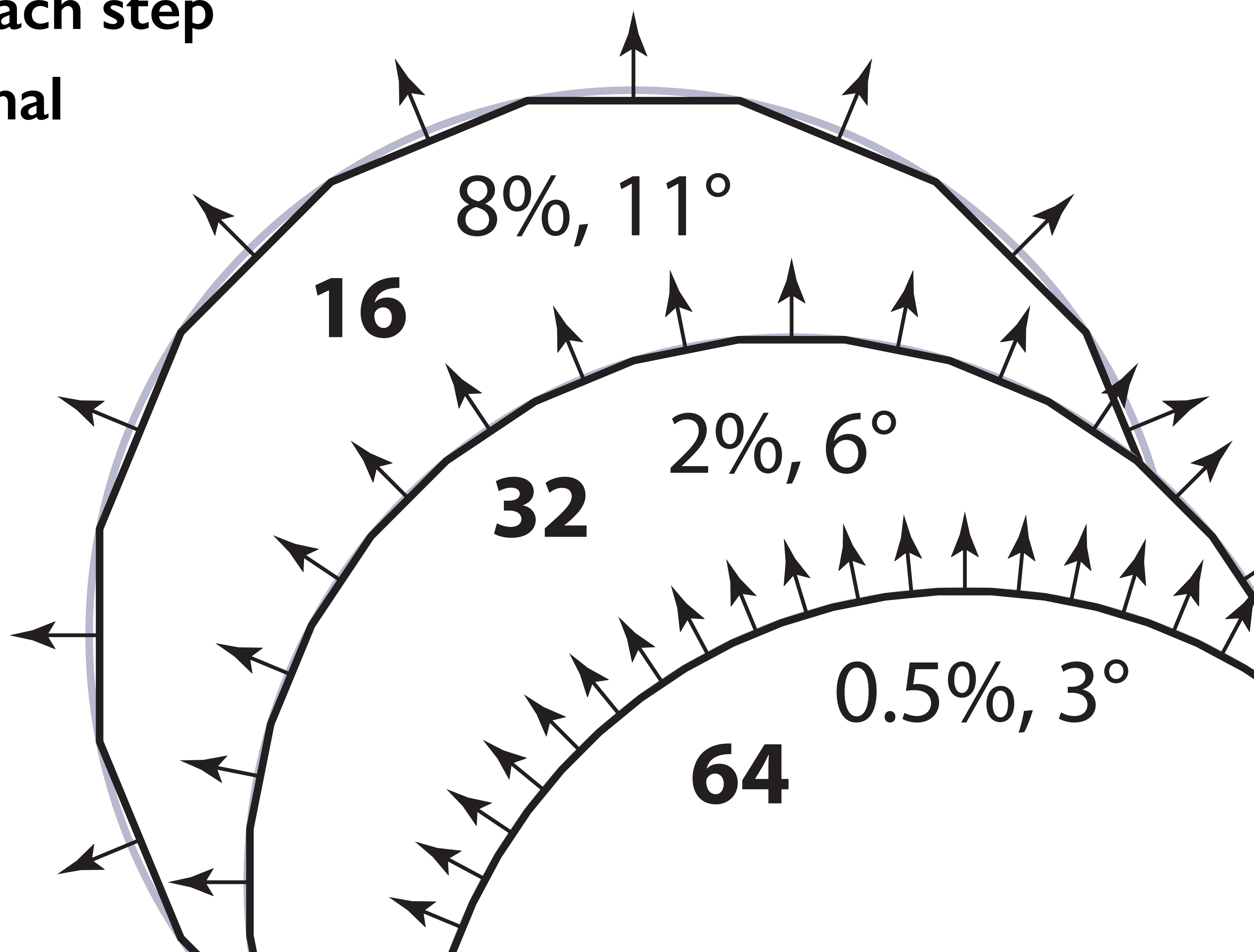
interpolated normals

How to think about vertex normals

- **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  - for mathematicians: error is $O(h^2)$
- **But the surface normals don't converge so well**
  - normal is constant over each triangle, with discontinuous jumps across edges
  - for mathematicians: error is only $O(h)$

Interpolated normals—2D example

- **Approximating circle with increasingly many segments**
- **Max error in position error drops by factor of 4 at each step**
- **Max error in normal only drops by factor of 2**



How to think about vertex normals

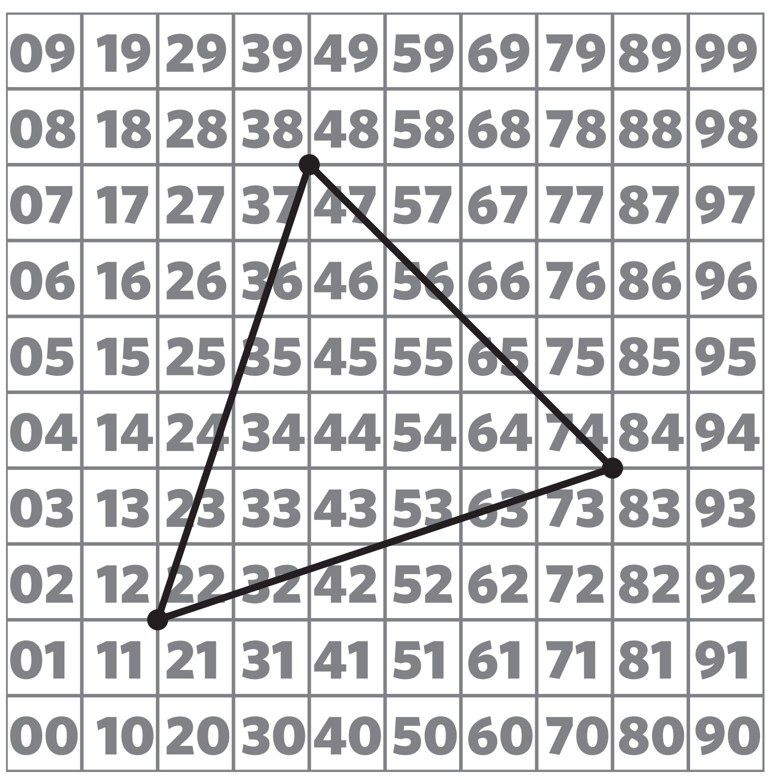
- **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  - for mathematicians: error is $O(h^2)$
- **But the surface normals don't converge so well**
  - normal is constant over each triangle, with discontinuous jumps across edges
  - for mathematicians: error is only $O(h)$

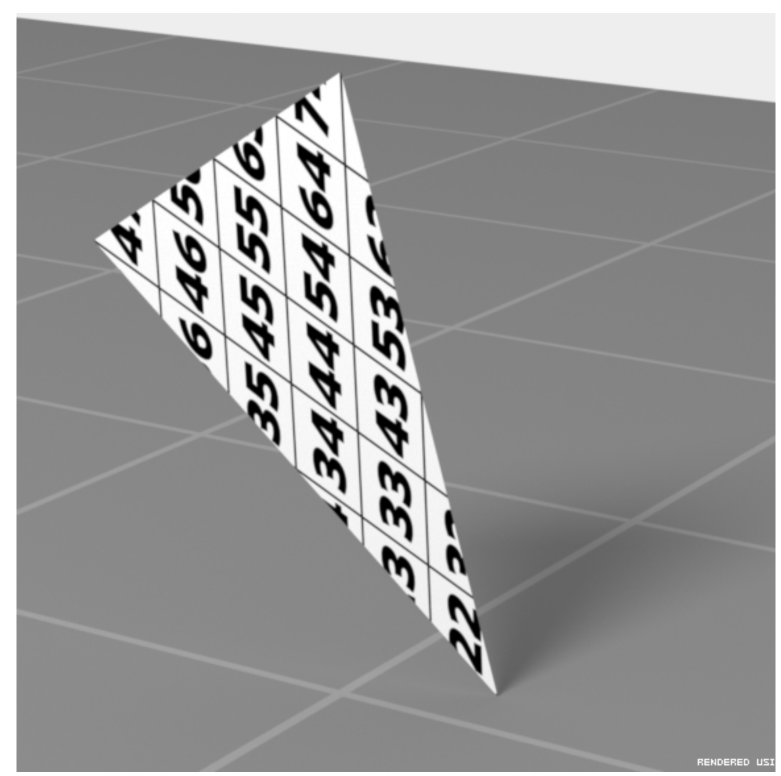
How to think about vertex normals

- **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  - for mathematicians: error is $O(h^2)$
- **But the surface normals don't converge so well**
  - normal is constant over each triangle, with discontinuous jumps across edges
  - for mathematicians: error is only $O(h)$
- **Better: store the "real" normal at each vertex, and *interpolate* to get normals that vary gradually across triangles**

Texture coordinates on meshes

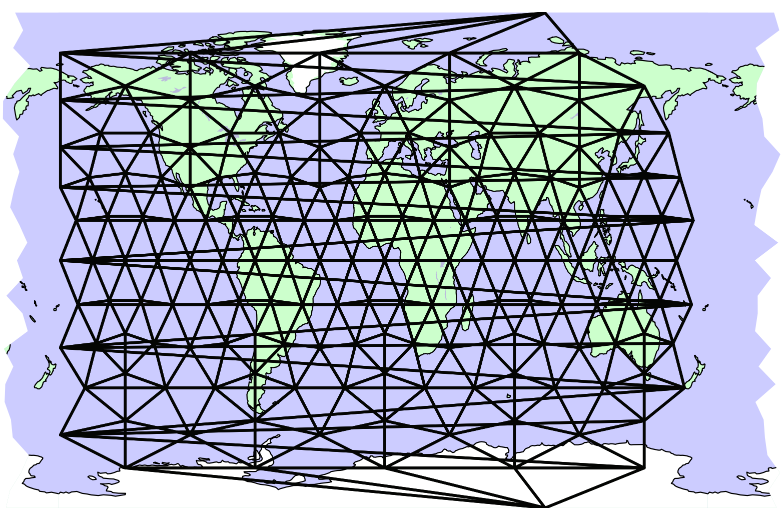
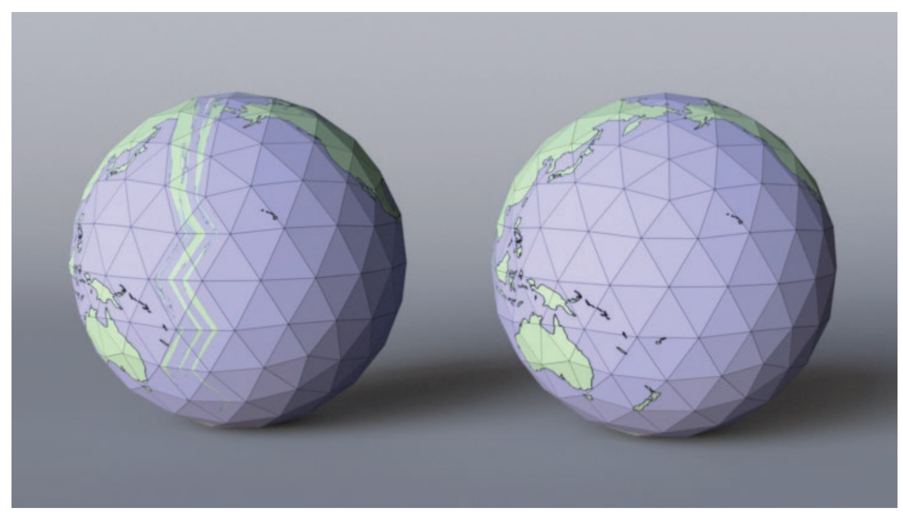
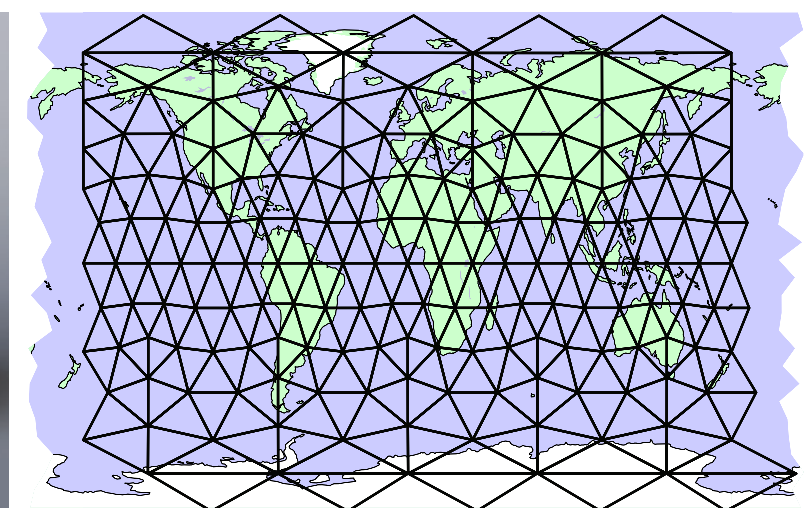
- **Texture coordinates are per-vertex data like vertex positions**
  – can think of them as a second position: each vertex has a position in 3D space and in 2D texture space
- **Interpolation defines (*u*,*v*)s for points inside triangles**





Discontinuities in texture coordinates

- **Sometimes textures are supposed to wrap around objects**
  - can do this with a tileable texture
  - texture coordinate needs to jump to the next repeat at some point

Texture coordinates at each vertex set to latitude and longitude. Mesh connected continuously everywhere. Result: triangles across longitude seam (international date line) stretch across the whole map.

Vertices near seam duplicated; copies have longitudes differing by 360 degrees. Mesh is not connected across this seam. Result: *u* coordinate discontinuous across seam, but due to texture wrapping, texture is continuous.

Practical encoding of mesh attributes

- **OBJ file format**
  - supports specifying normals and texture coordinates at vertices
  - particularly flexible about continuity of attributes
- **Demo**
  - simple file with one triangle
  - effects of normals and texture coords
  - exploration of continuity and discontinuity