Topics

1. Introduction: What is Computer Graphics?
2. Raster Images (image input/output devices and representation)
3. Scan conversion (pixels, lines, triangles)
4. Ray Casting (camera, visibility, normals, lighting, Phong illumination)
5. Ray Tracing (shadows, supersampling, global illumination)
6. Spatial Data Structures (AABB trees, OBB, bounding spheres, octree)
7. Meshes (connectivity, smooth interpolation, uv-textures, subdivision, Laplacian smoothing)
8. 2D/3D Transformations (Translate, Rotate, Scale, Affine, Homography, Homogeneous coordinates)
9. Viewing and Projection (matrix composition, perspective, Z-buffer)
10. Shader Pipeline (Graphics Processing Unit)
11. Animation (kinematics, keyframing, Catmull-Romm interpolation, physical simulation)
12. 3D curves and objects (Hermite, Bezier, cubic curves, curve continuity, extrusion/revolve surfaces)
13. Advanced topics overview

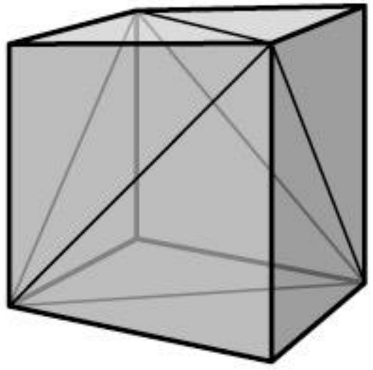
Topic 7.

Meshes

*some slides adapted from Steve Marschner

Ottawa Convention Center

A small triangle mesh

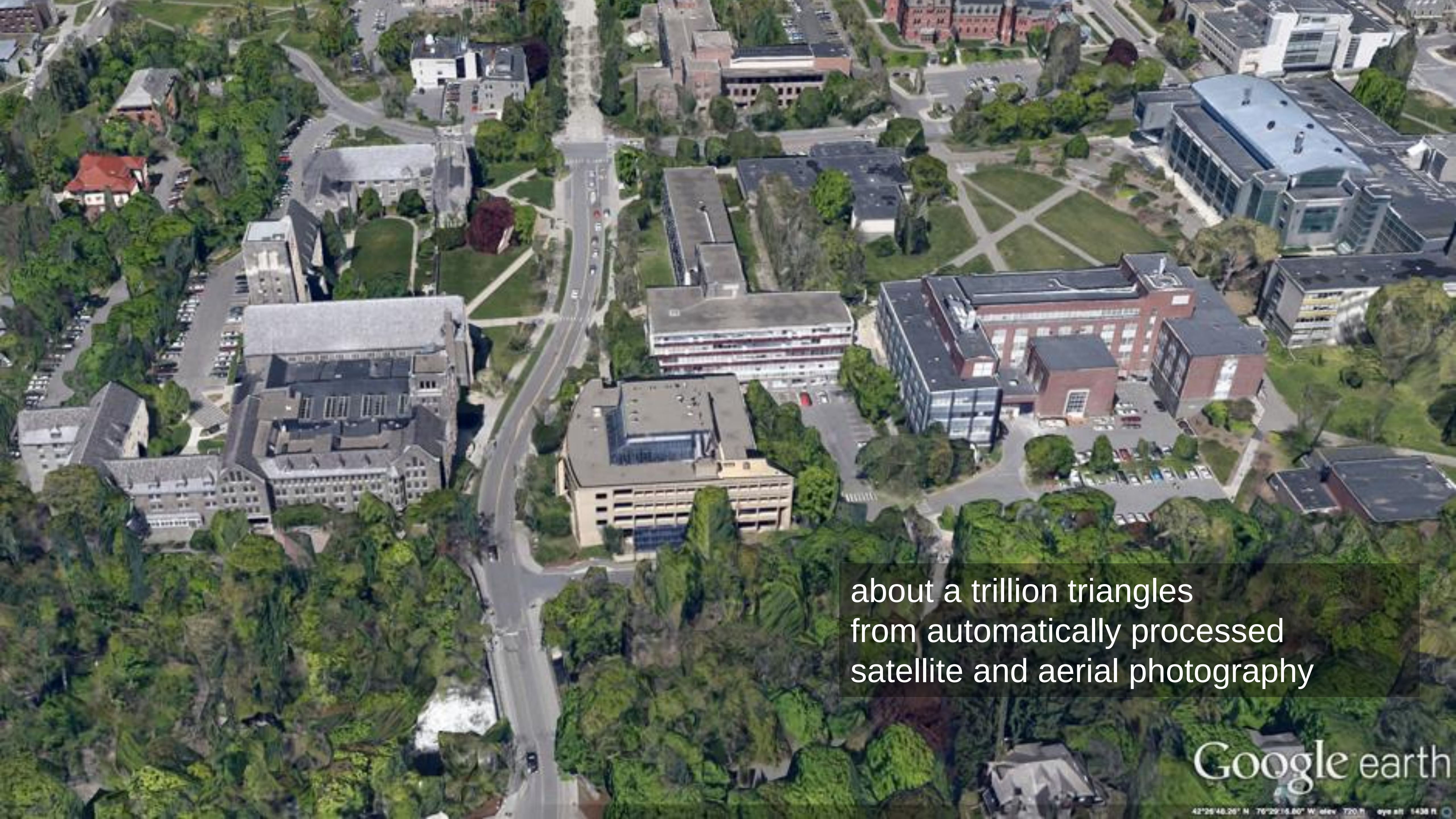


12 triangles, 8 vertices

A large triangle mesh

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

10 million triangles from a high-resolution 3D scan


about a trillion triangles
from automatically processed
satellite and aerial photography
Google earth

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

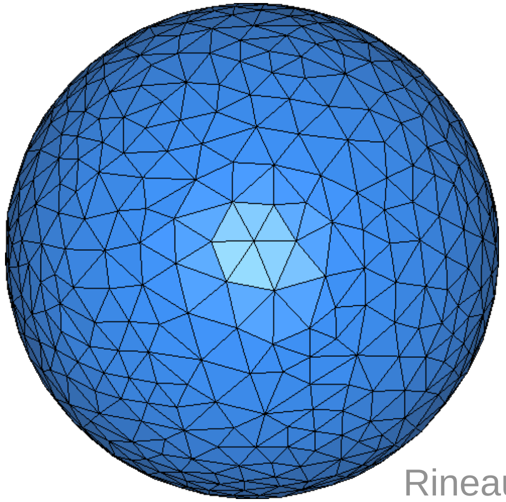
Geometrically, a mesh is a piecewise planar surface

- 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

Andrzej Barabasz



Rineau & Yvinec CGAL manual

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
- applications

  finding triangle strips

  computing subdivision surfaces

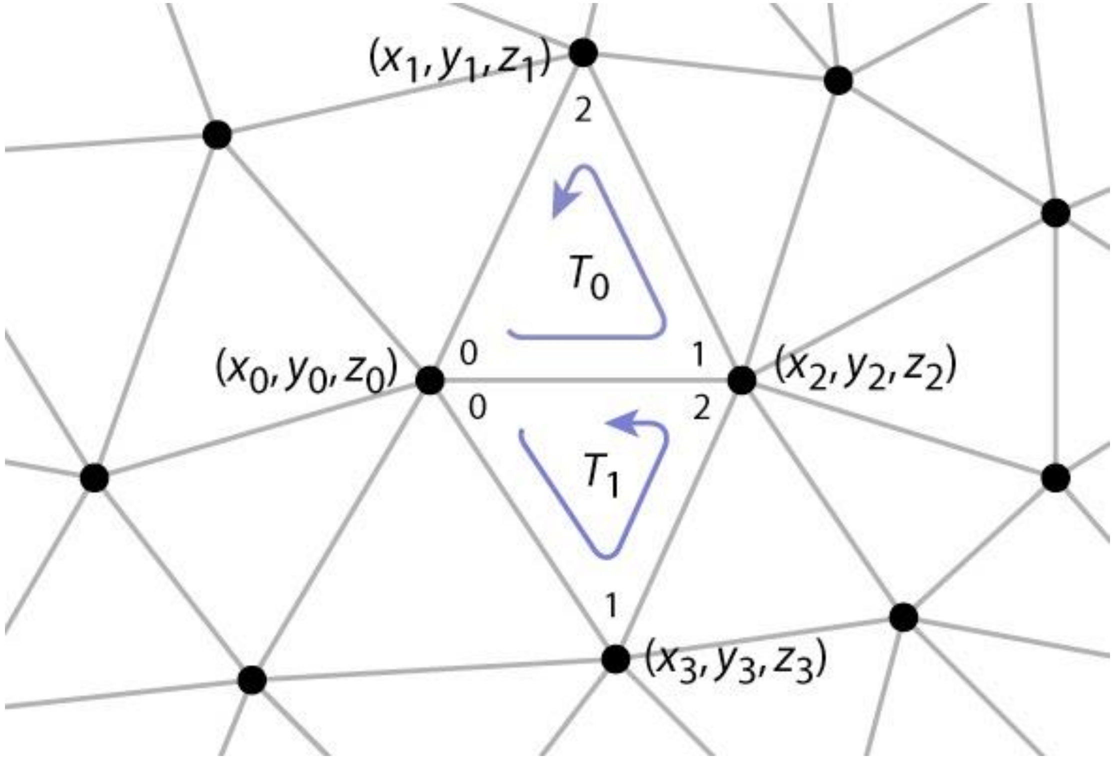
  mesh editing

Representations for triangle meshes

- Separate triangles
- Indexed triangle set
  - Shared vertices
- Triangle strips and triangle fans
  - Fast transmission
- Triangle-neighbor data structure
  - supports adjacency queries
- Winged-edge data structure
  - supports general polygon meshes

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
	⋮	⋮	⋮



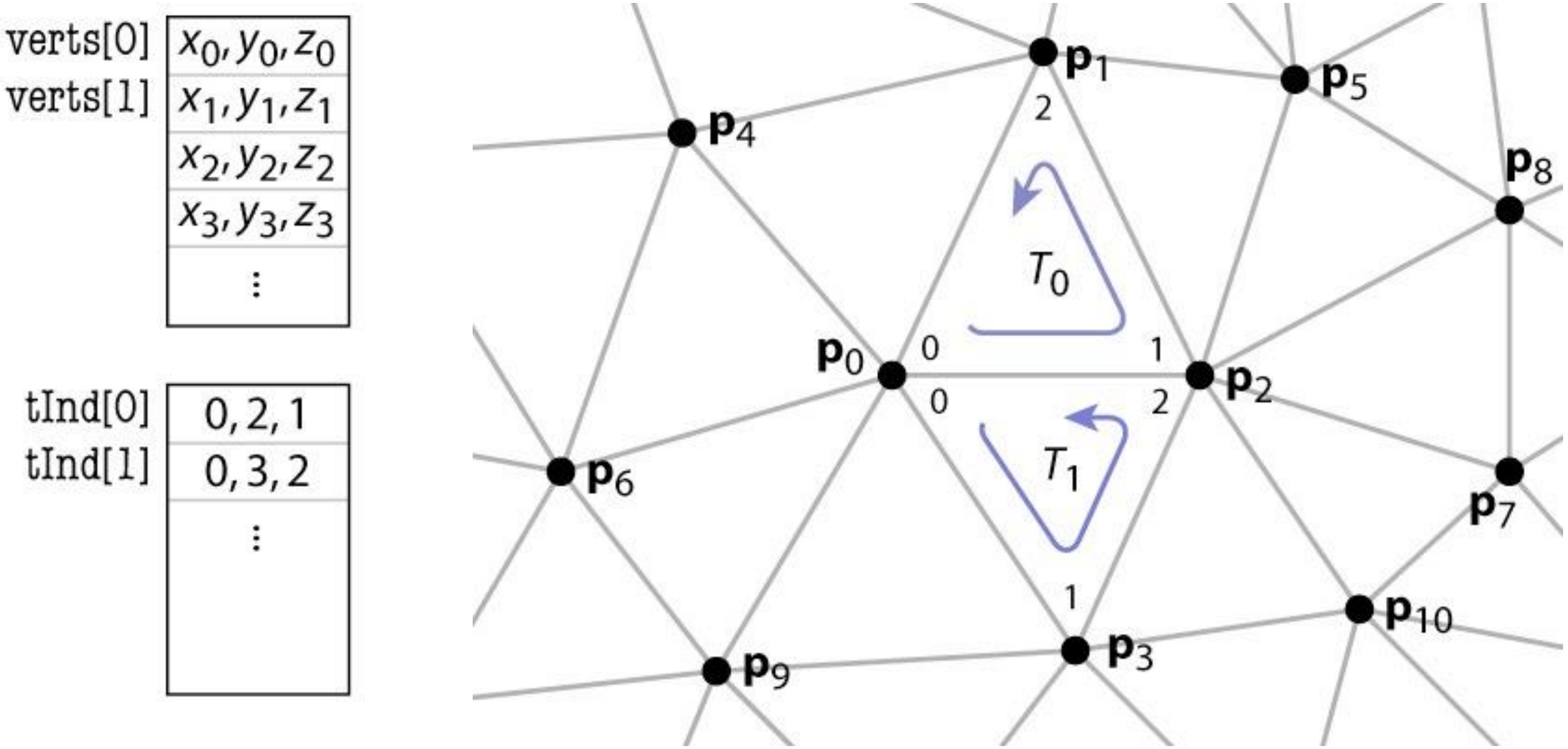
Separate triangles

array of triples of points: float[*n*][3][3] for n triangles

various problems

- wastes space (each vertex stored multiple times)
- cracks due to roundoff
- difficulty of finding neighbors at all

Indexed triangle set

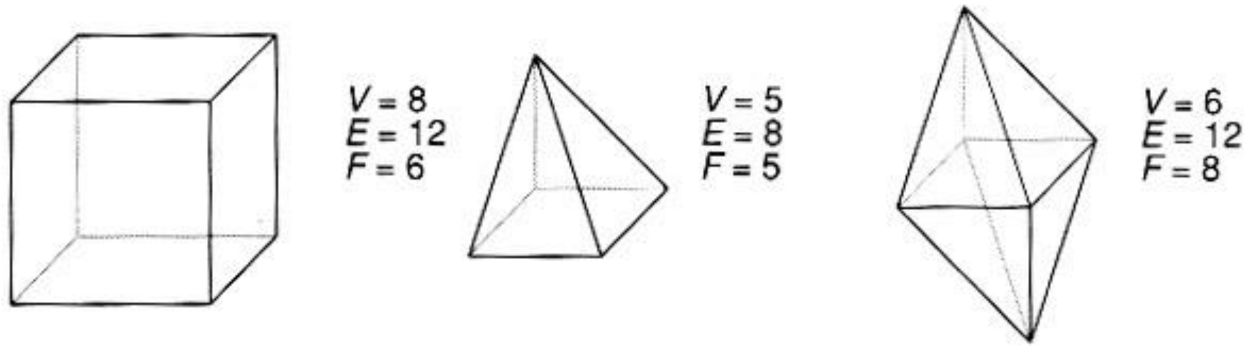


Eulers Formula

n_T = #tris; n_V = #verts; n_E = #edges

$n_V - n_E + n_T = 2$ for a simple closed surface

- and in general sums to small integer
- For triangle mesh $3*n_T = 2*n_E$ Why?
- n_T:n_E:n_V is about 2:3:1



Indexed triangle set

array of vertex positions

- float[n_V][3]

array of triples of indices (per triangle)

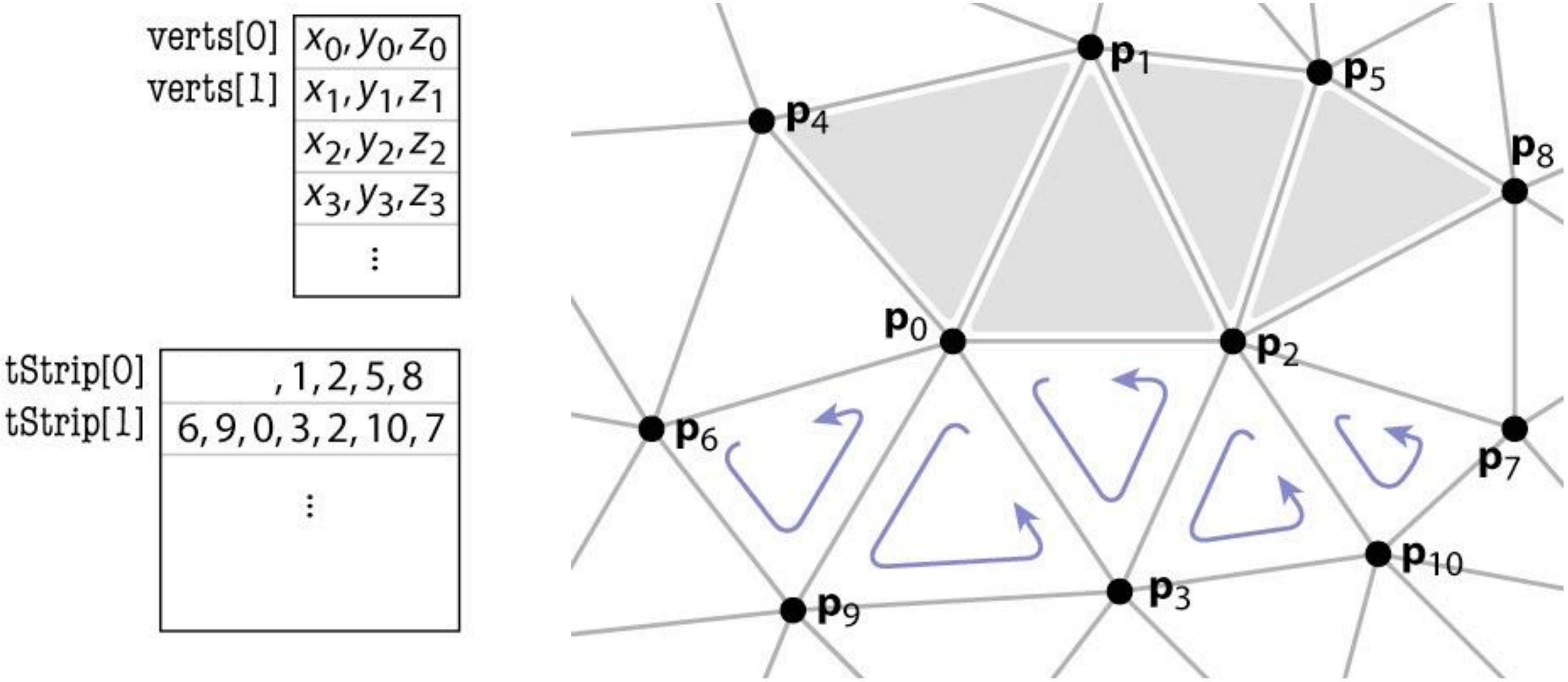
- int[n_T][3]

represents topology and geometry separately.

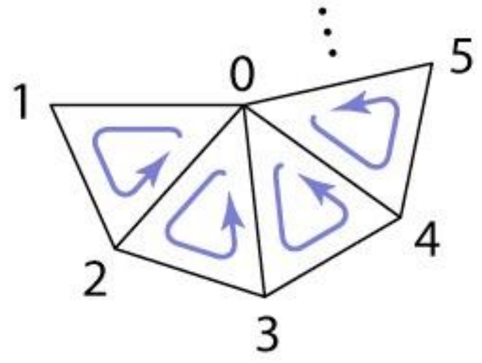
finding neighbors is at least well defined

How can one create an vertex -> triangle adjacency list?

Triangle strips and fans



strip



fan

Triangle neighbor structure

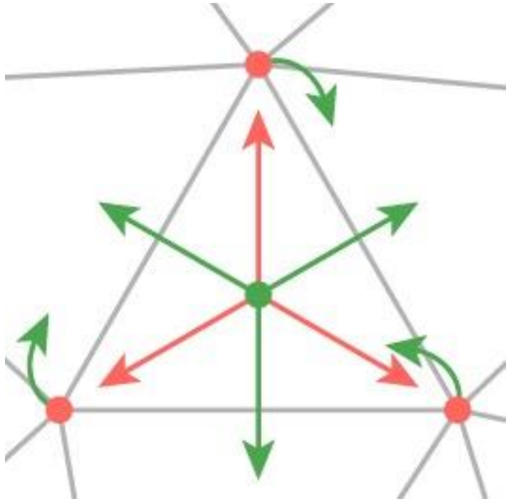
```
Triangle {
        Triangle nbr[3];
        Vertex vertex[3];
        }

// t.nbr[i] is adjacent
// across the edge from i to i+1

Vertex {
        // ... per-vertex data ...
        Triangle t;  // any adjacent tri
        }

// ... or ...

Mesh {
        // ... per-vertex data ...
        int tInd[nt][3];  // vertex indices
        int tNbr[nt][3]; // indices of neighbor triangles
        int vTri[nv]; // index of any adjacent triangle
        }
```



Winged-edge mesh

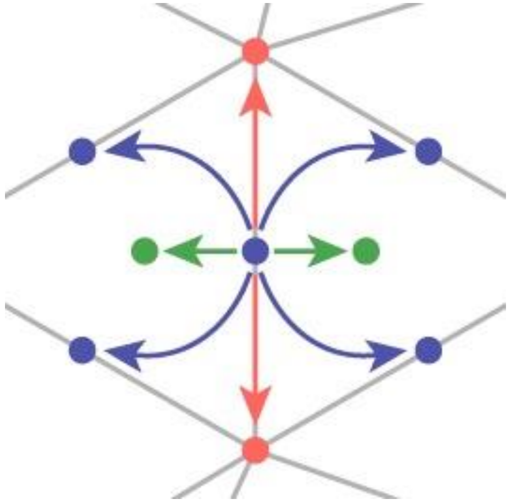
Edge-centric rather than
face-centric

- therefore also works for polygon meshes

Each (oriented) edge points to:

- left and right forward edges
- left and right backward edges
- front and back vertices
- left and right faces

Each face or vertex points to
one edge



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

Texture coordinates

- 2D coordinates that tell you how to paste images on the surface

Positions

- at some level this is just another piece of data
- position varies continuously between vertices

REMEMBER Barycentric co-ordinates for interpolation!

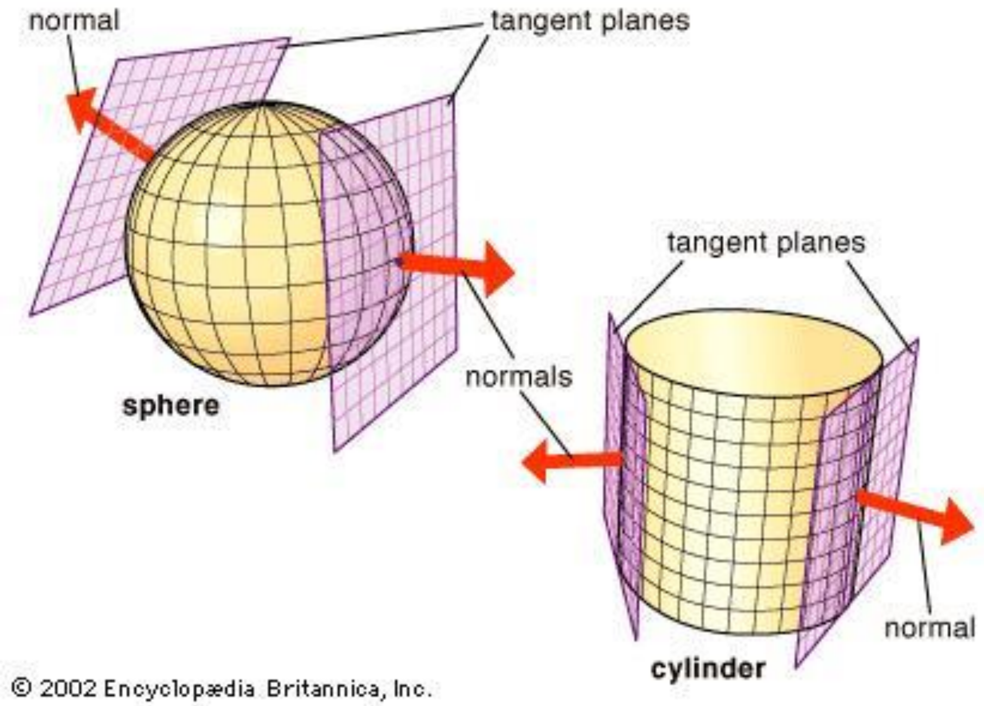
Normal Vectors

Tangent plane

- at a point on a smooth surface in 3D, there is a unique plane tangent to the surface, called the tangent plane

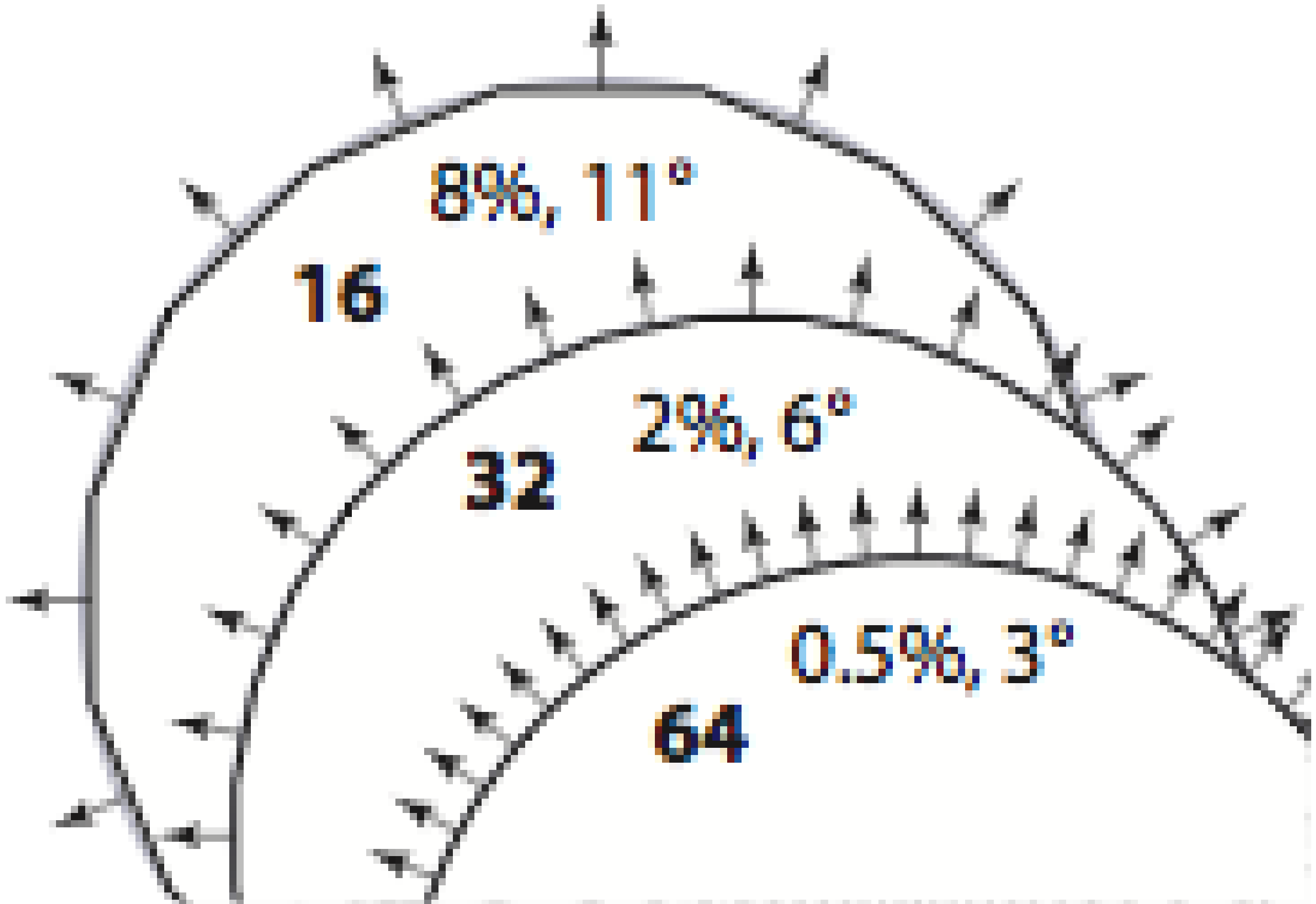
Normal vector

- vector perpendicular to a surface (that is, to the tangent plane)
- only unique for smooth surfaces (not at corners, edges)



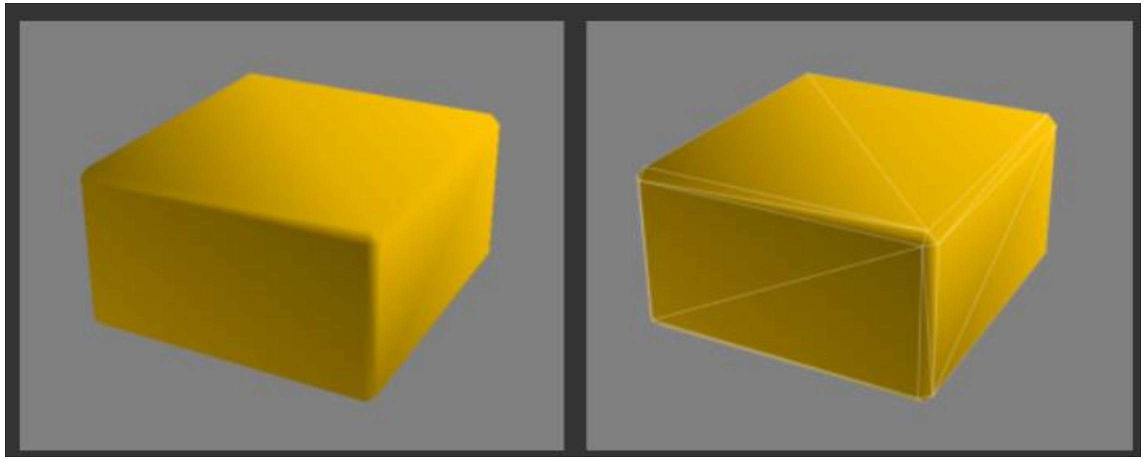
Interpolated normals—2D example

Approximating circle with increasingly many segments



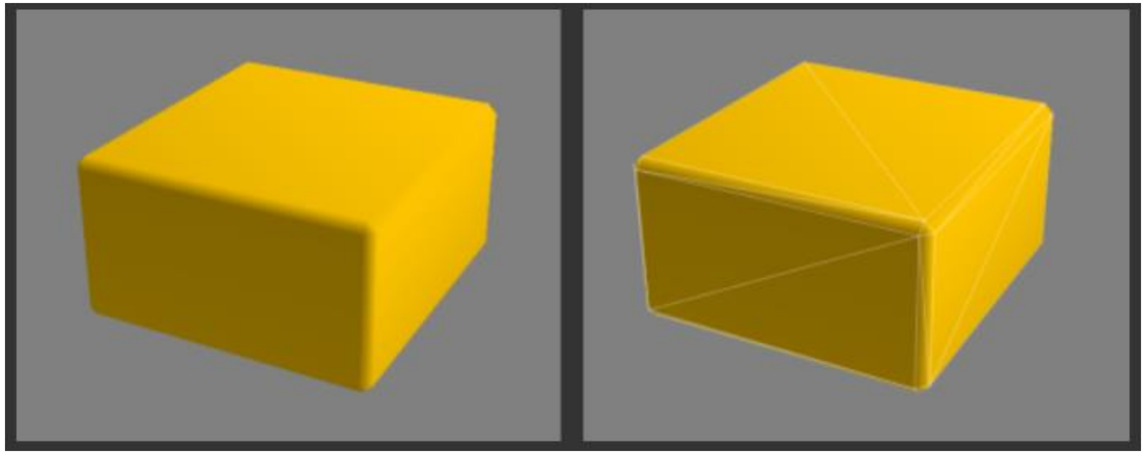
Computing a per-vertex normal

Average the per-face normals

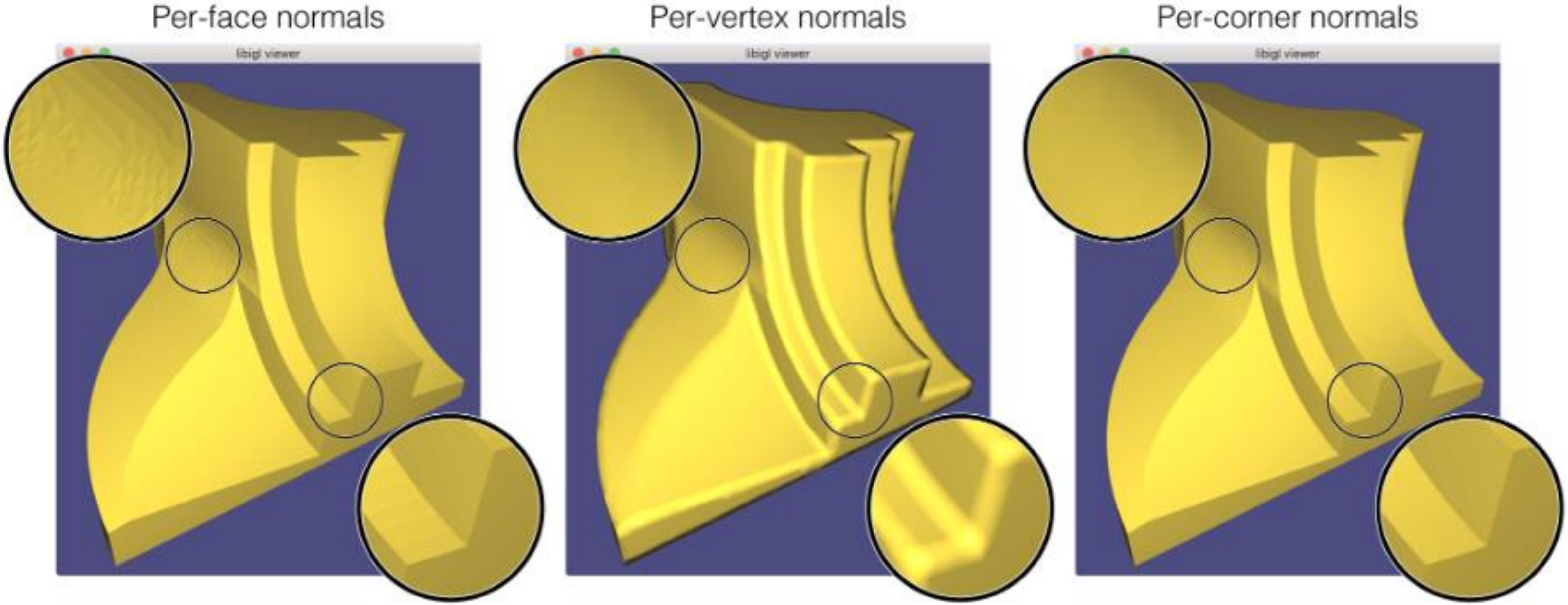


Angle weighted per-face normals

Area-weighted per-face normals



Computing a per-vertex normal for sharp edges



Surface parameterization

A surface in 3D is a two-dimensional thing

Sometimes we need 2D coordinates for points on the surface

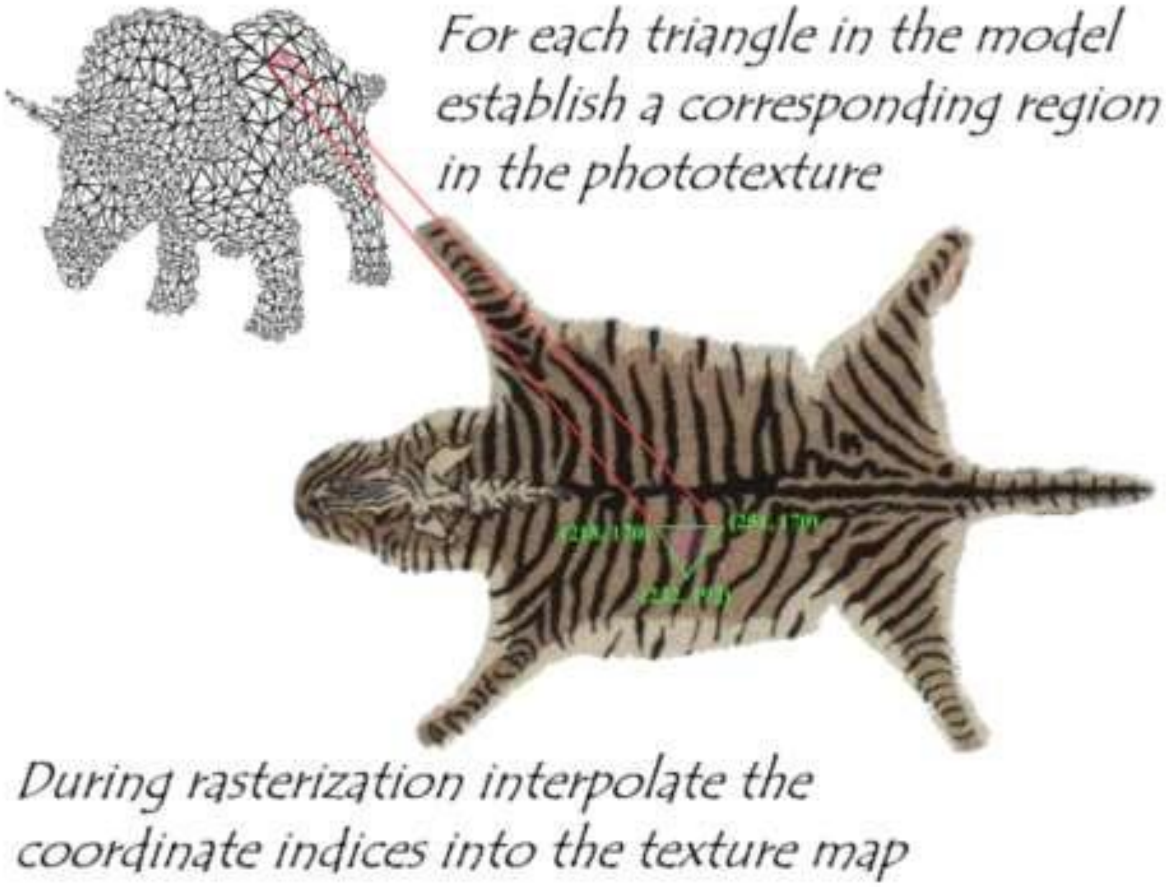
Defining these coordinates is *parameterizing* the surface

Examples:

- cartesian coordinates on a rectangle (or other planar shape)
- cylindrical coordinates (θ, y) on a cylinder
- latitude and longitude on the Earth's surface
- spherical coordinates (θ, ϕ) on a sphere

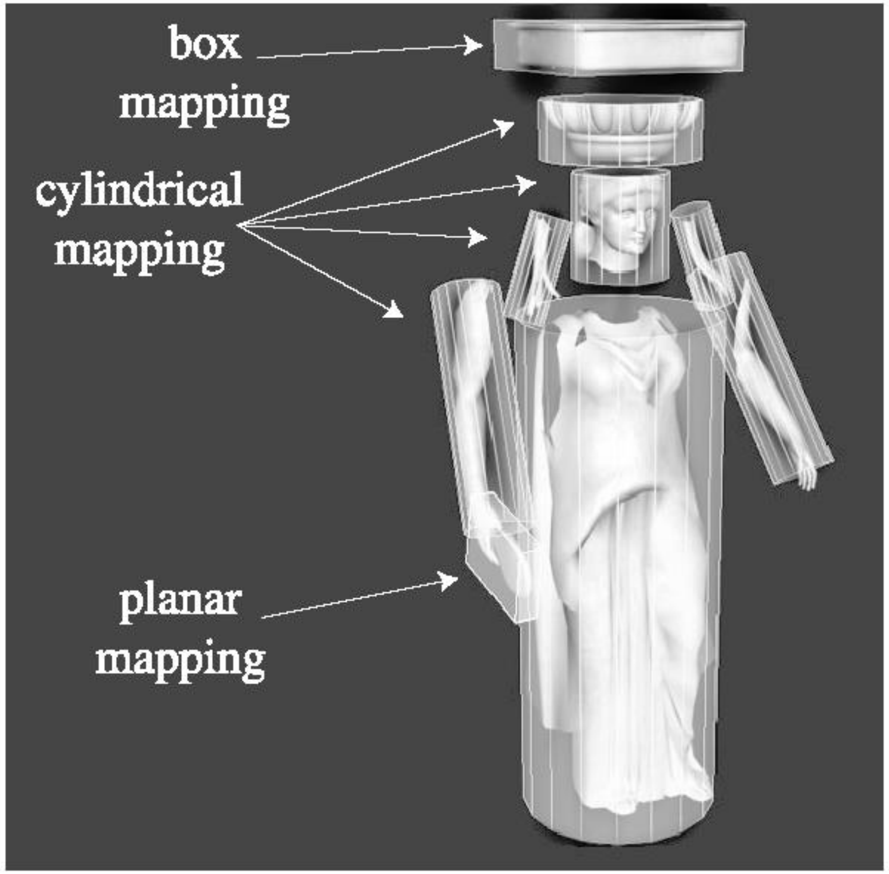
Texture coordinates

How does one establish correspondence? (UV mapping)



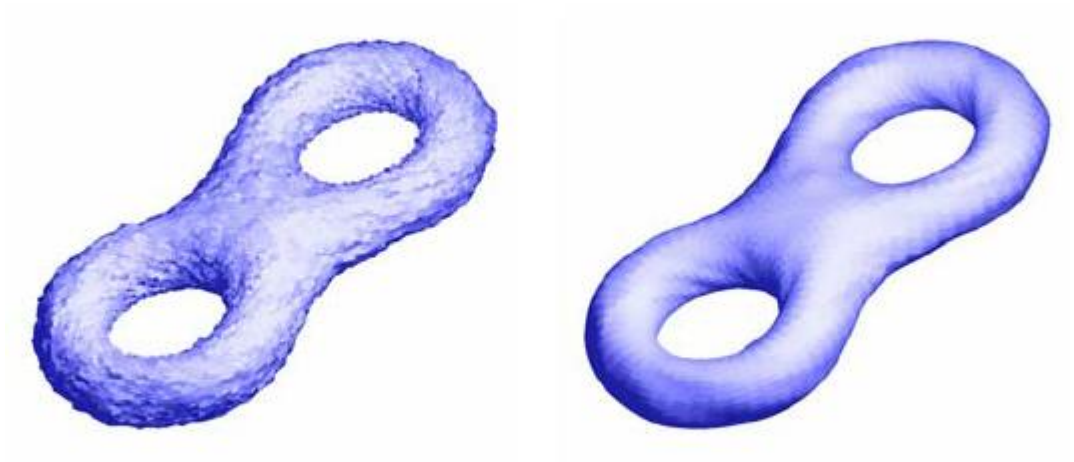
Examples of coordinate functions

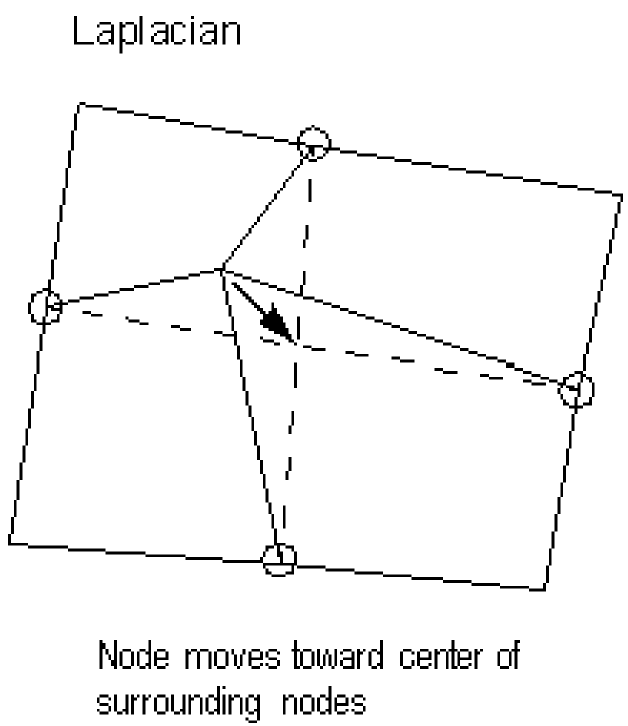
Complex surfaces: project parts to parametric surfaces



Neighbour Averaging: Laplacian smoothing

Move each vertex towards the average of its neighbours.



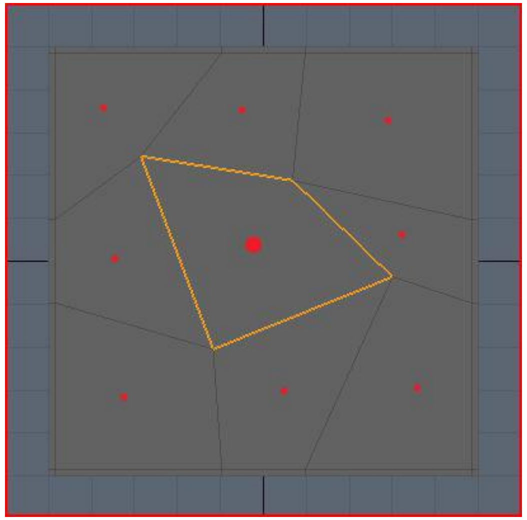


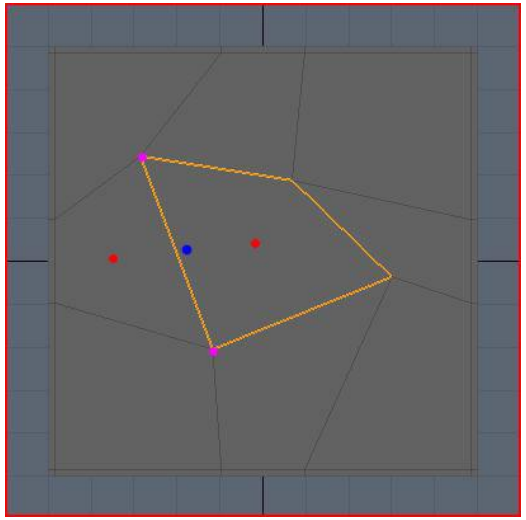
Catmull-Clark

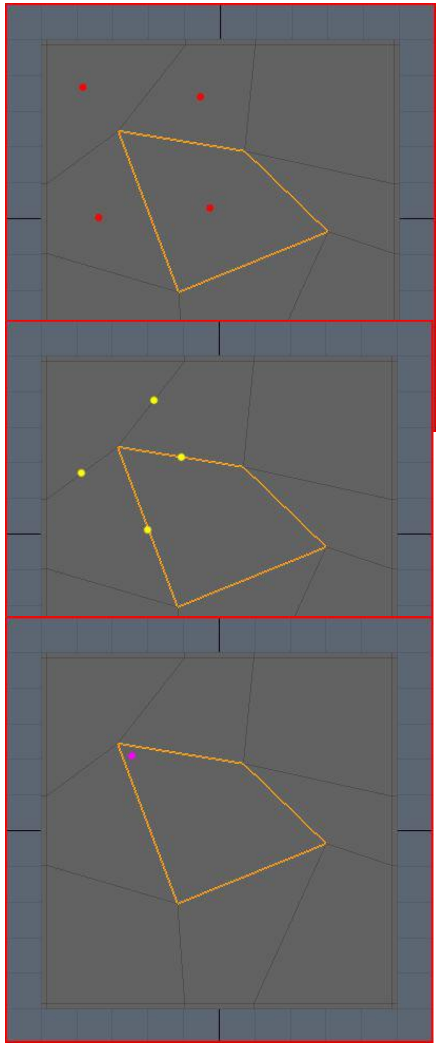
1. Add a new point to each face (V/n), called the face-point.
2. Add a new point to each edge $(F/2n) + (V/2n)$, called the edge-point.
3. Move the vertex to another position, called the vertex-point.

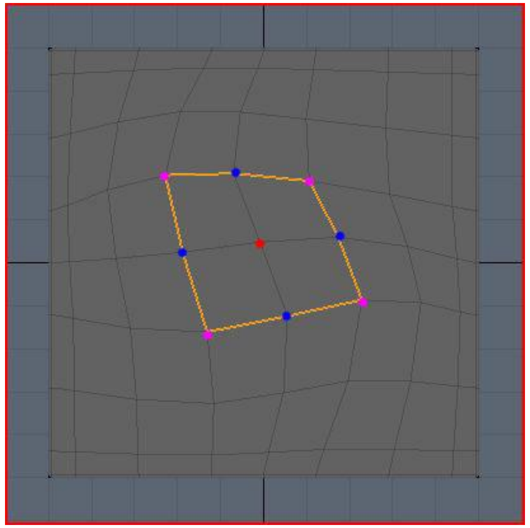
$$(F/n) + (2E/n) + (V(n-3)/n)$$

4. Connect the new points.









Catmull-Clark in action