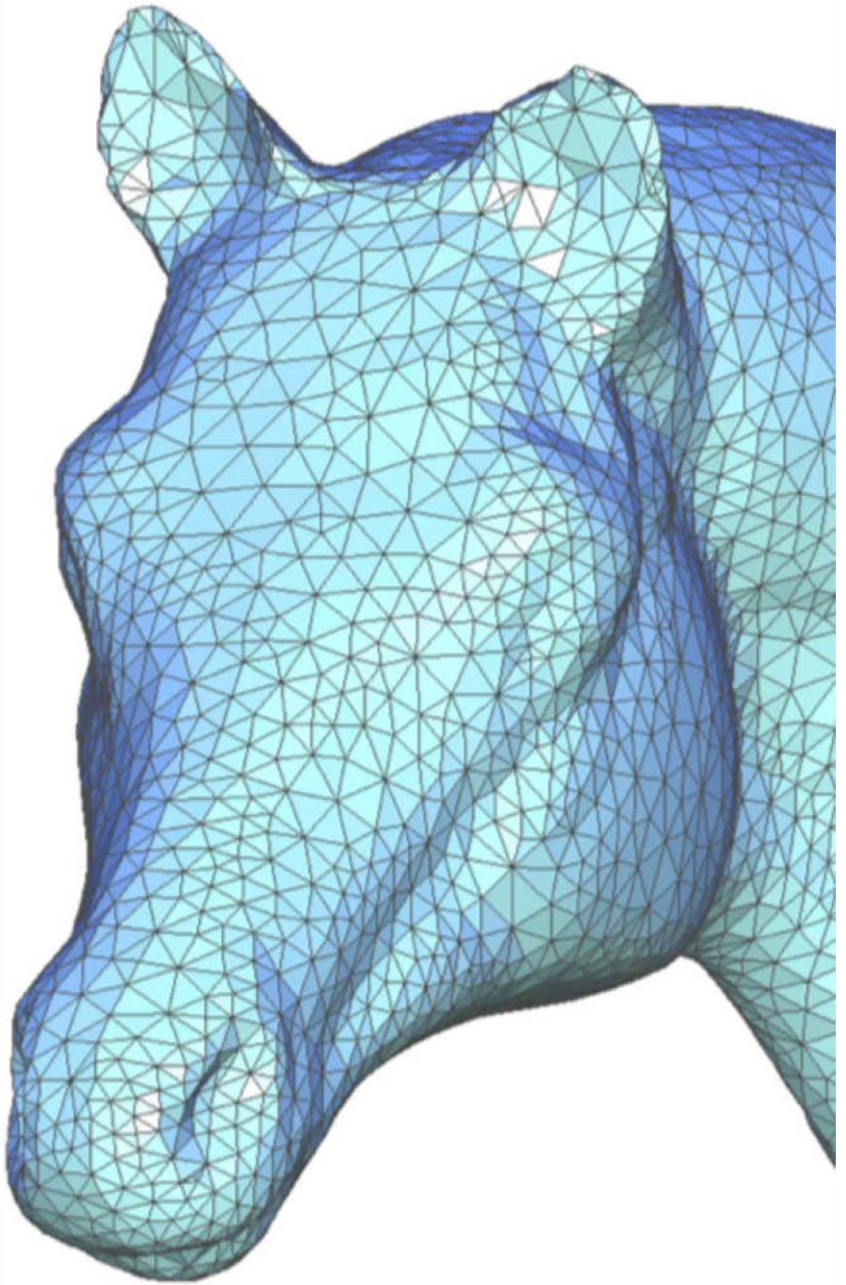
Meshes

Ottawa Convention Center



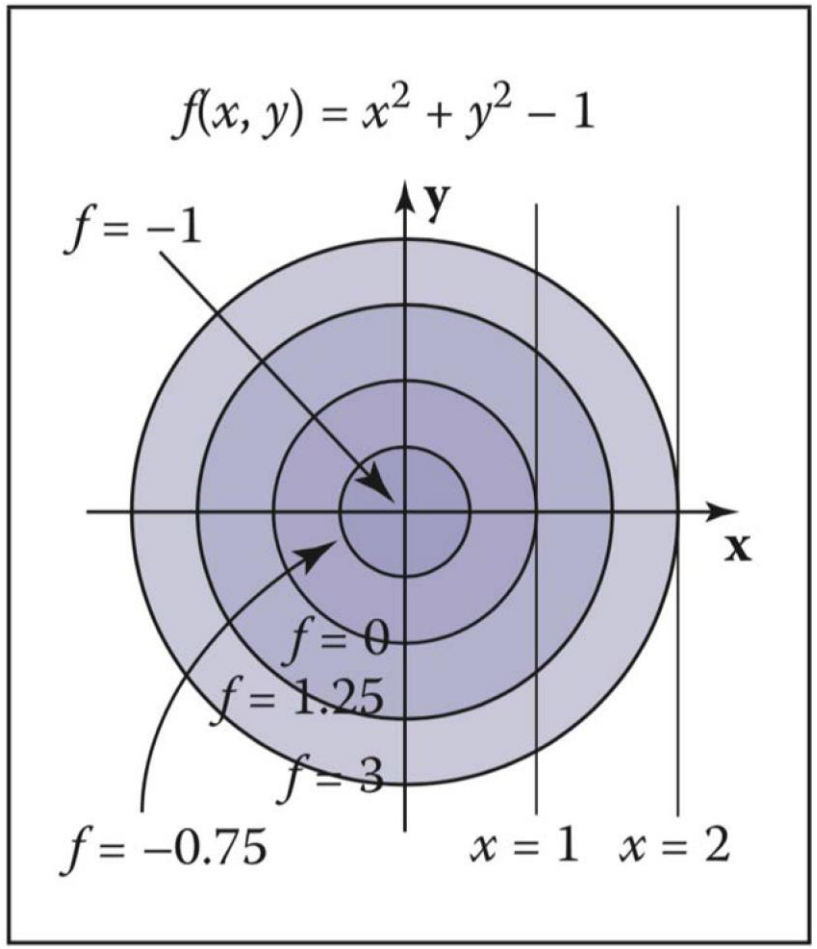
Some Slides/Images adapted from Marschner and Shirley and David Levin

Agenda

- Types of Surfaces
- Data Structures for Triangle Meshes
- Normals for Meshes
- Texture Mapping
- Subdivision Surfaces

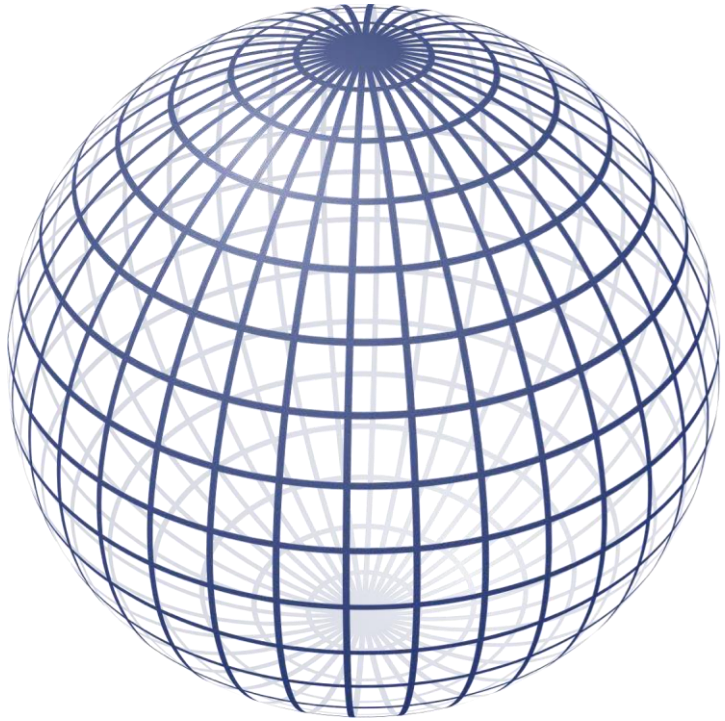
Surface Representations in Graphics

Implicit Surface

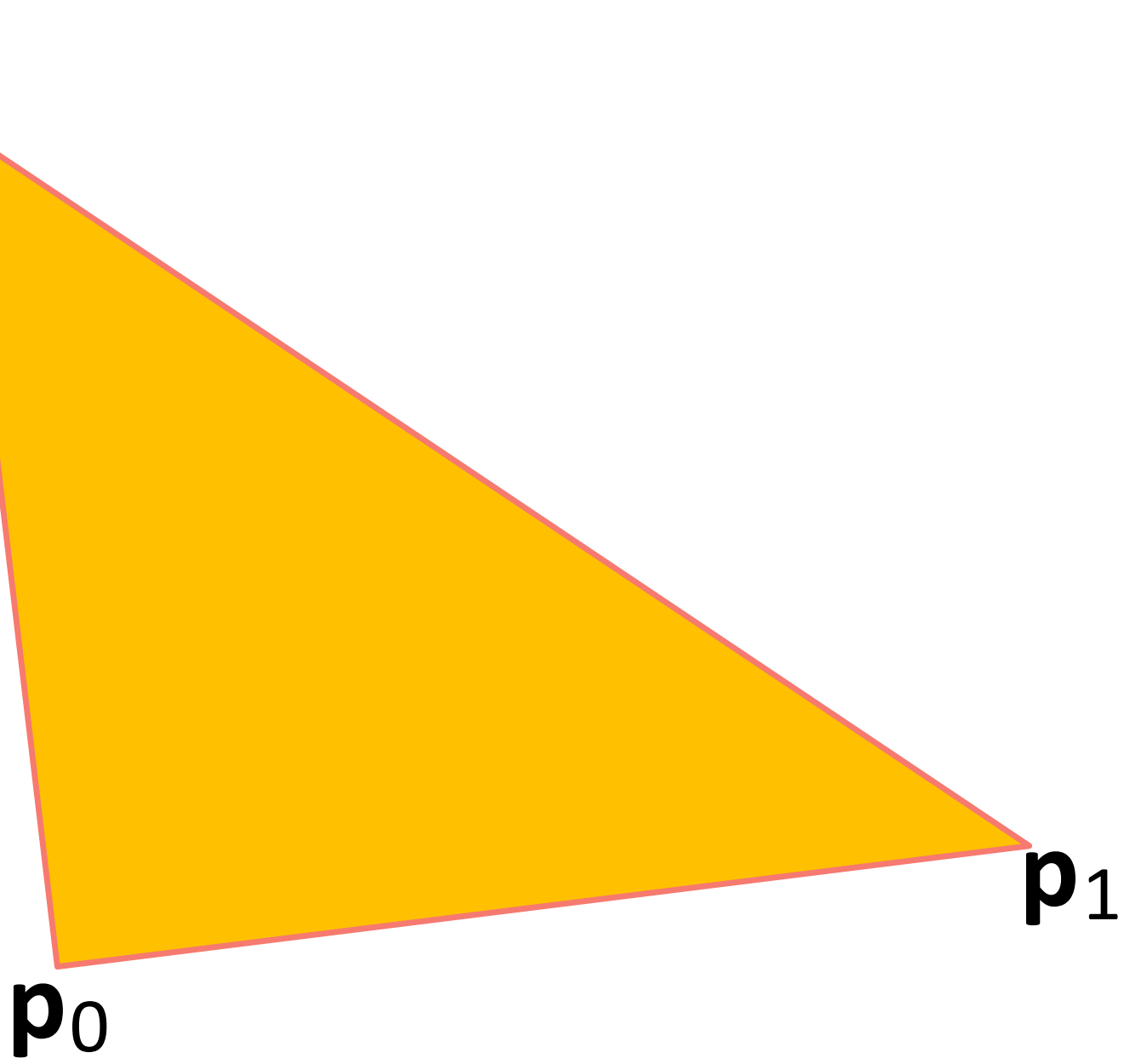


Parametric Surface

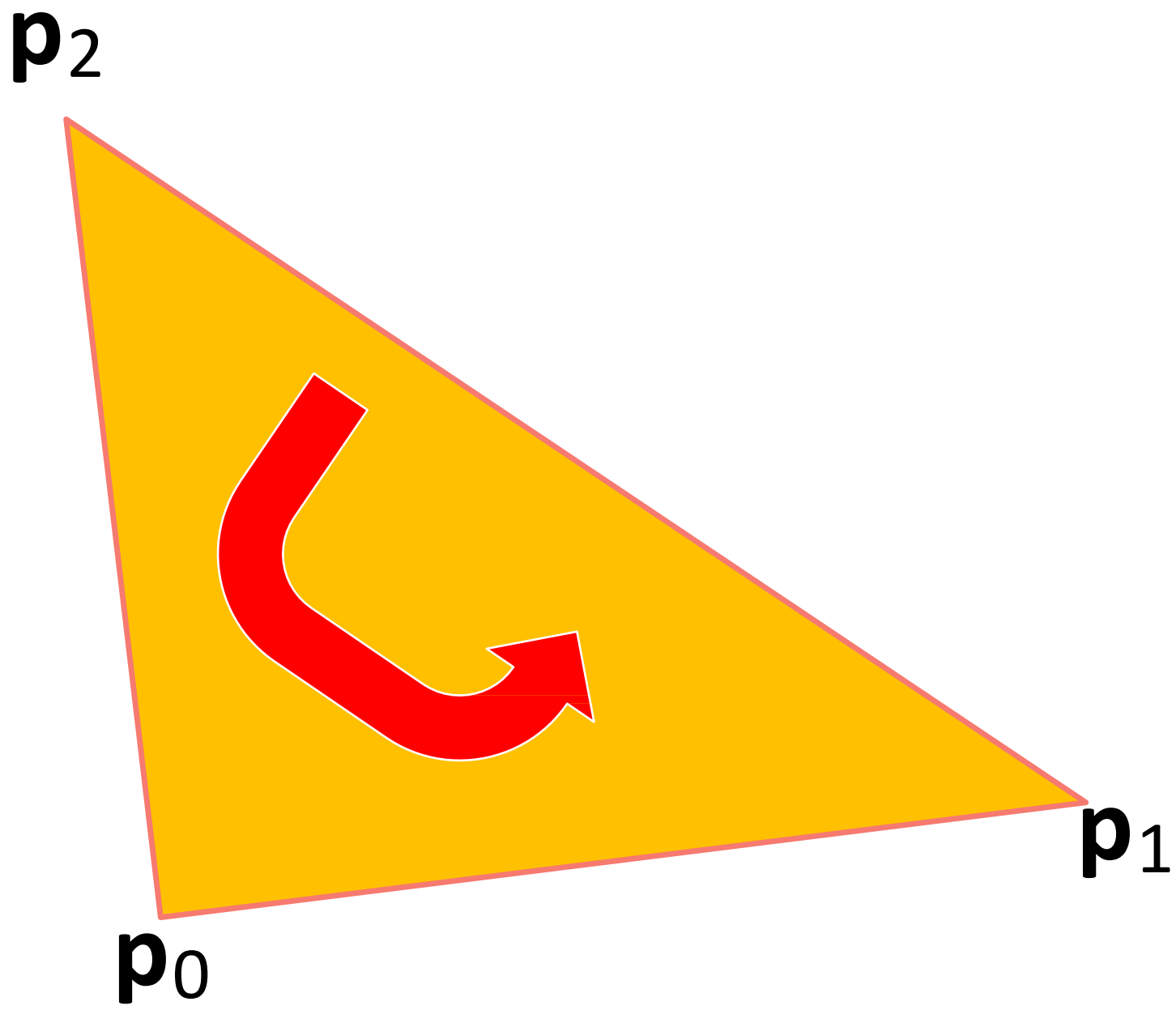
$$x = r \cos\phi \sin\theta,$$
$$y = r \sin\phi \sin\theta,$$
$$z = r \cos\theta.$$



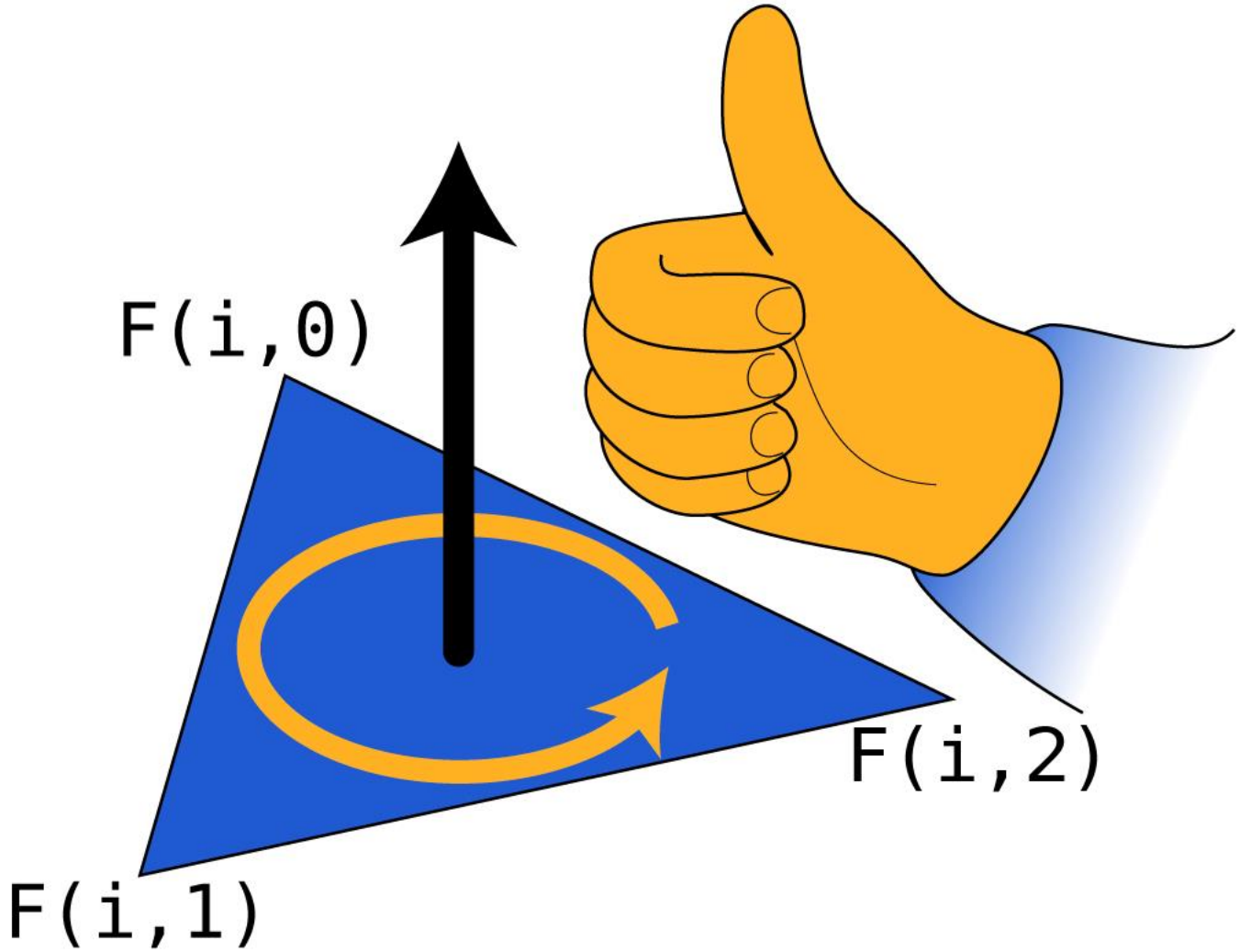
Triangles



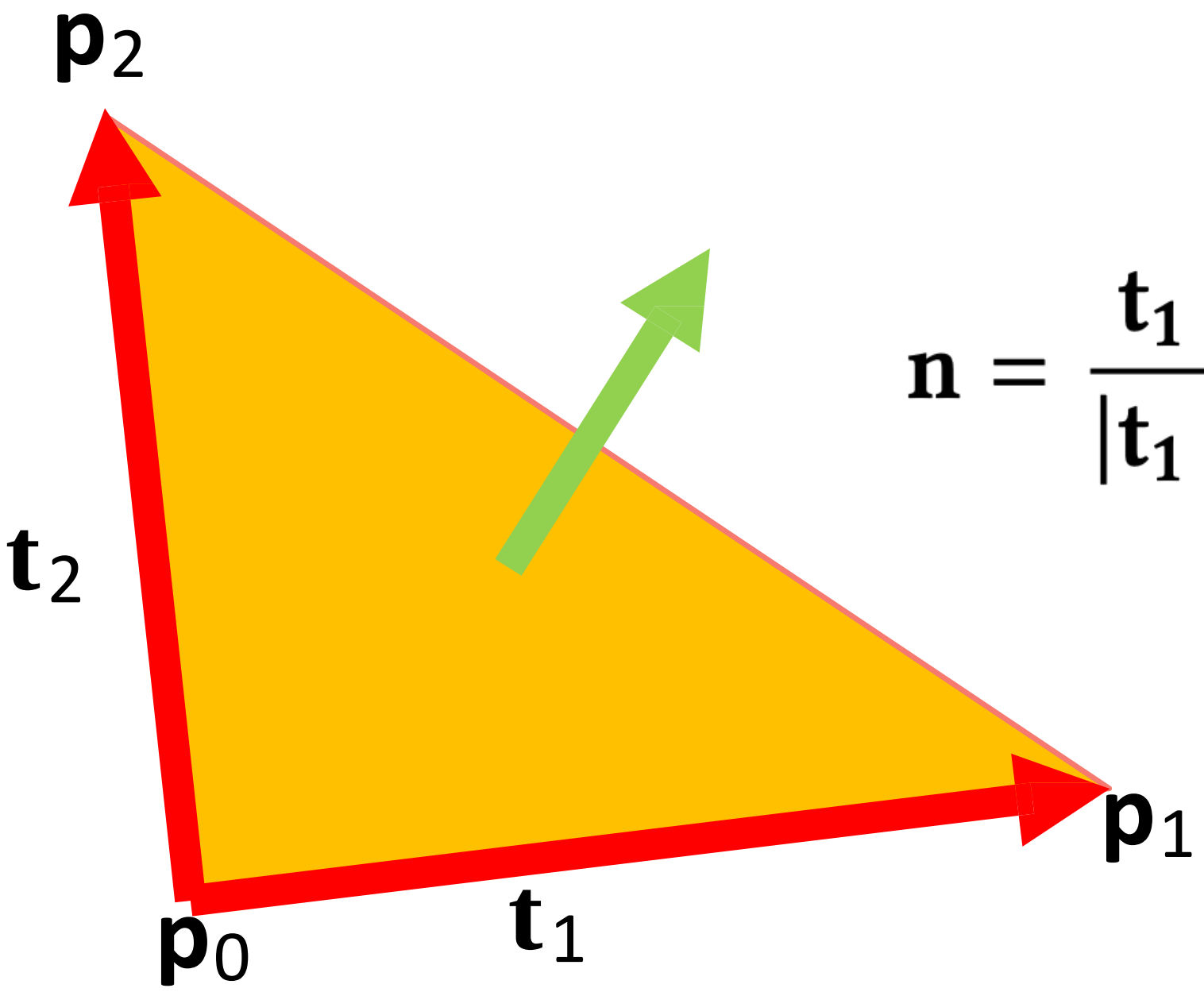
Triangles



Triangles



Triangles

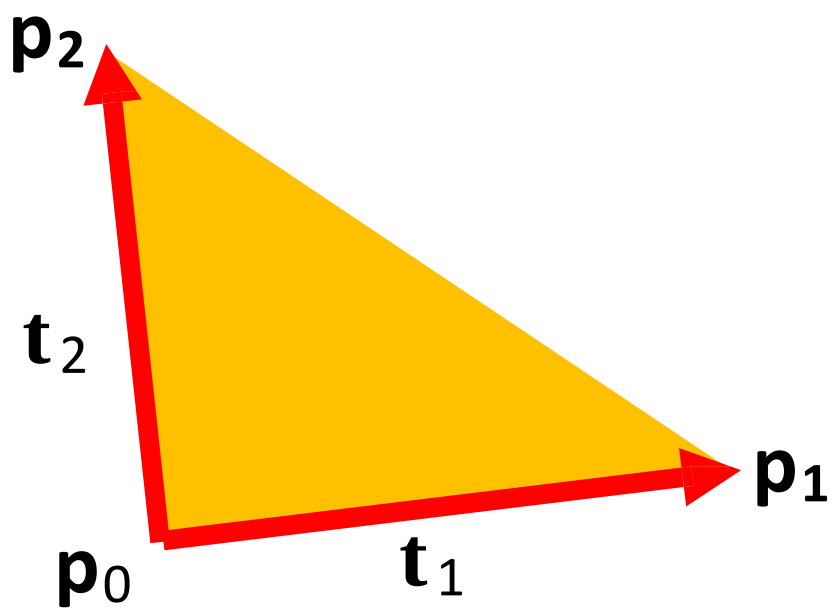


$$\mathbf{n} = \frac{\mathbf{t}_1 \times \mathbf{t}_2}{|\mathbf{t}_1 \times \mathbf{t}_2|}$$

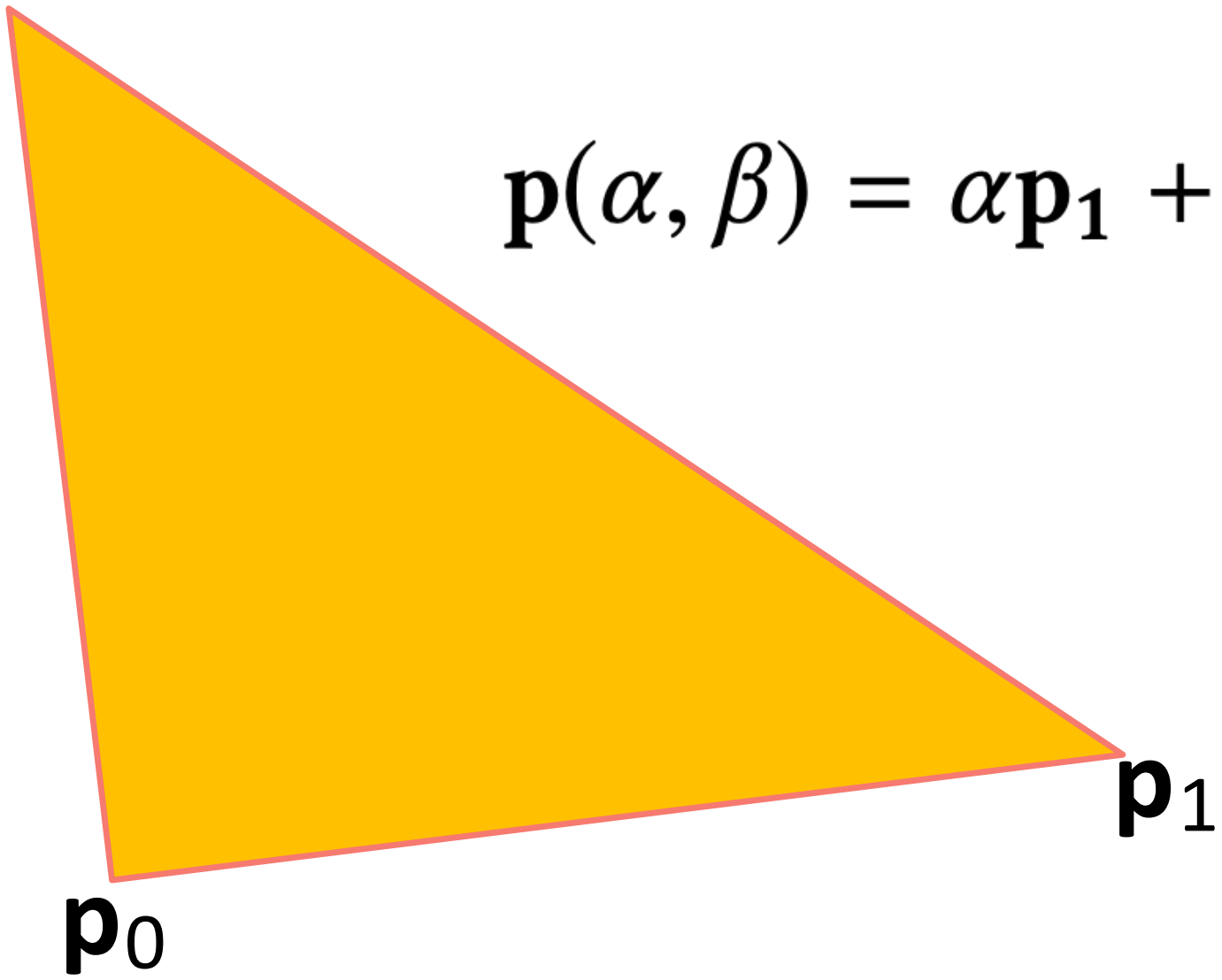
Barycentric Coordinates

$$\mathbf{p}(\alpha, \beta) = \mathbf{p_0} + \alpha(\mathbf{p_1} - \mathbf{p_0}) + \beta(\mathbf{p_2} - \mathbf{p_0})$$

$$\mathbf{p}(\alpha, \beta) = \alpha\mathbf{p_1} + \beta\mathbf{p_2} + (1 - \alpha - \beta)\mathbf{p_0}$$



Barycentric Coordinates



$$\mathbf{p}(\alpha, \beta) = \alpha\mathbf{p}_1 + \beta\mathbf{p}_2 + (1 - \alpha - \beta)\mathbf{p}_0$$

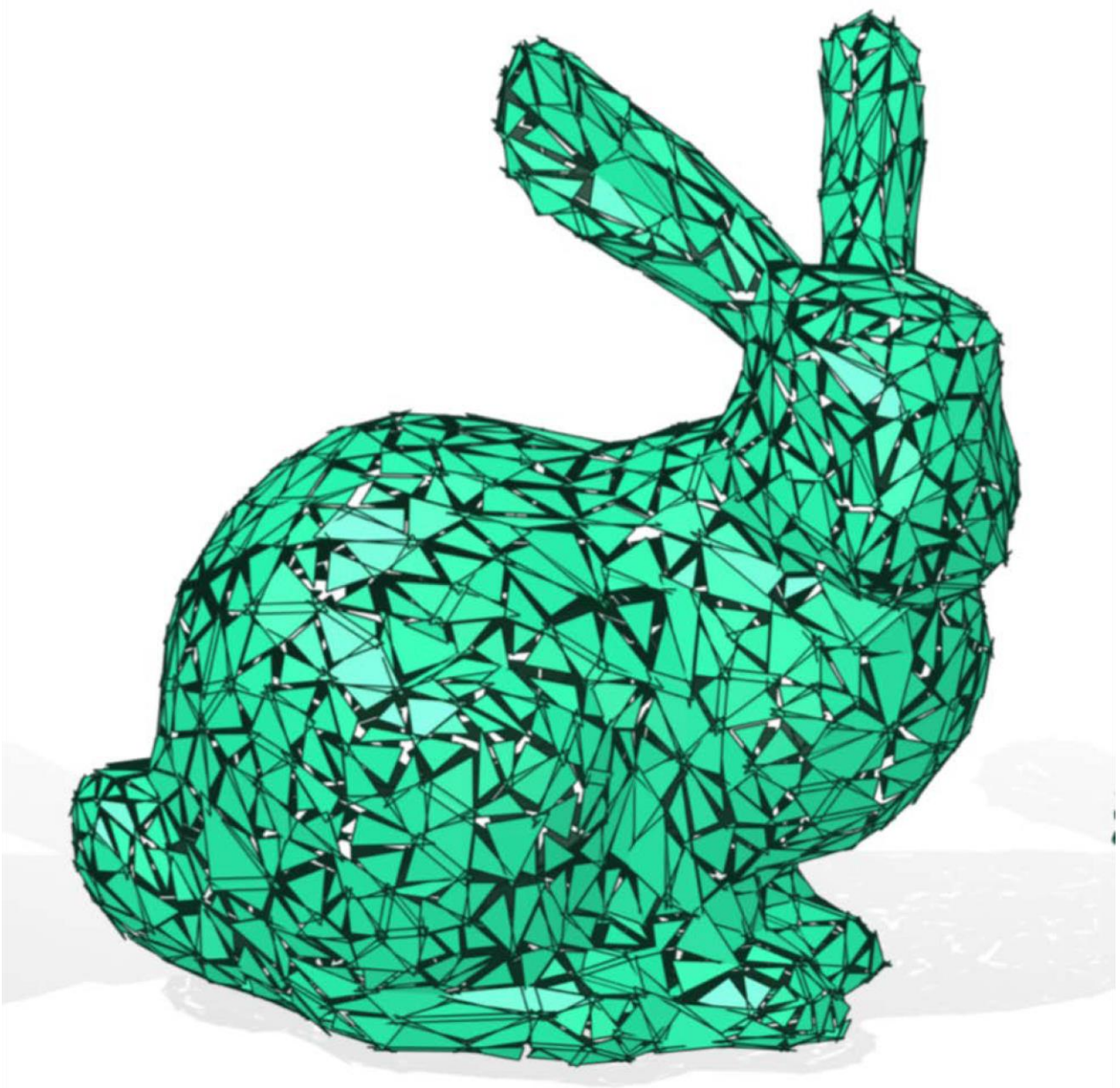
$$\alpha \geq 0$$

$$\beta \geq 0$$

$$\alpha + \beta \leq 1$$

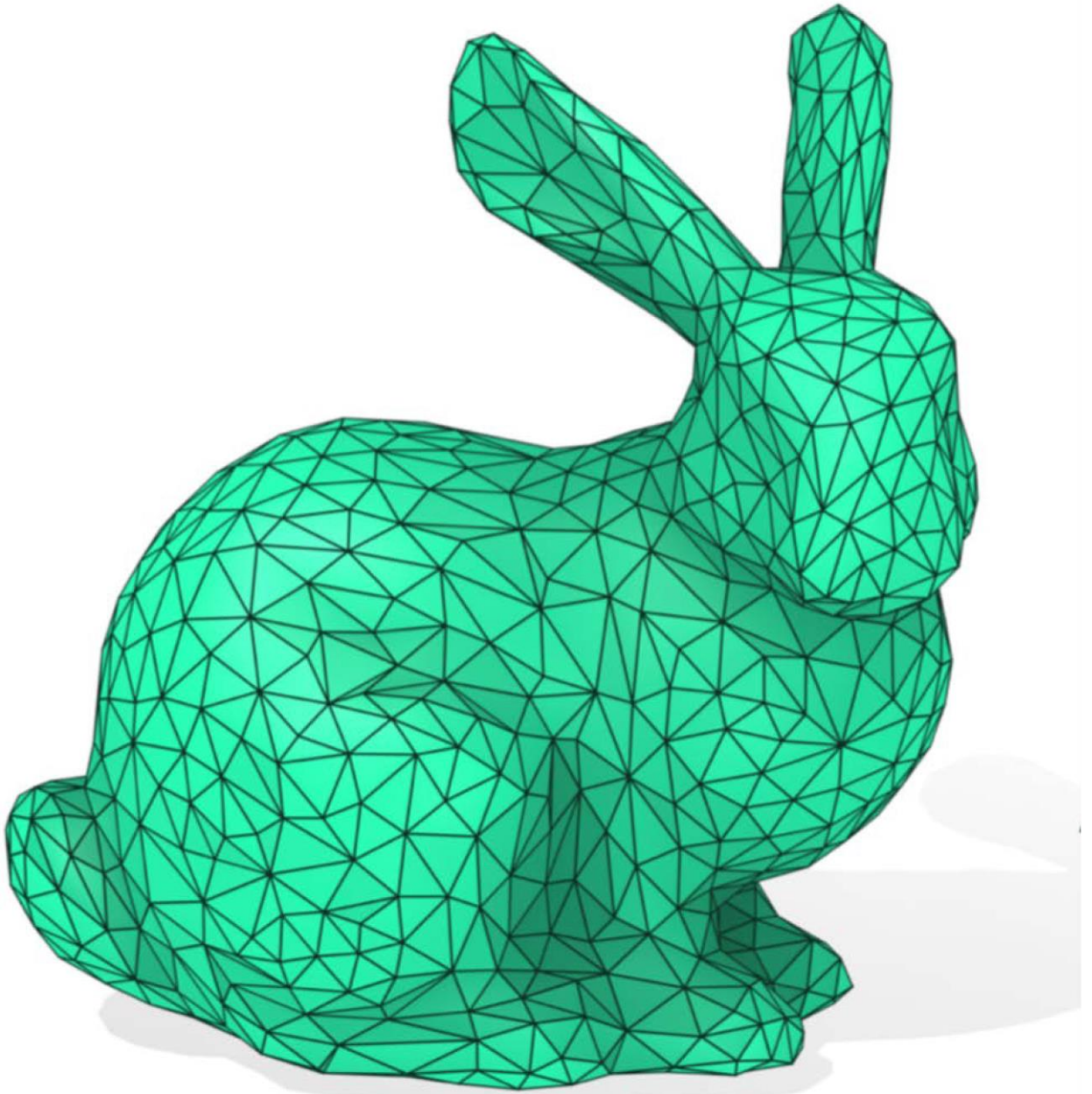
$$\gamma = 1 - \alpha - \beta$$

Triangle Soup



Triangle Mesh

Soup

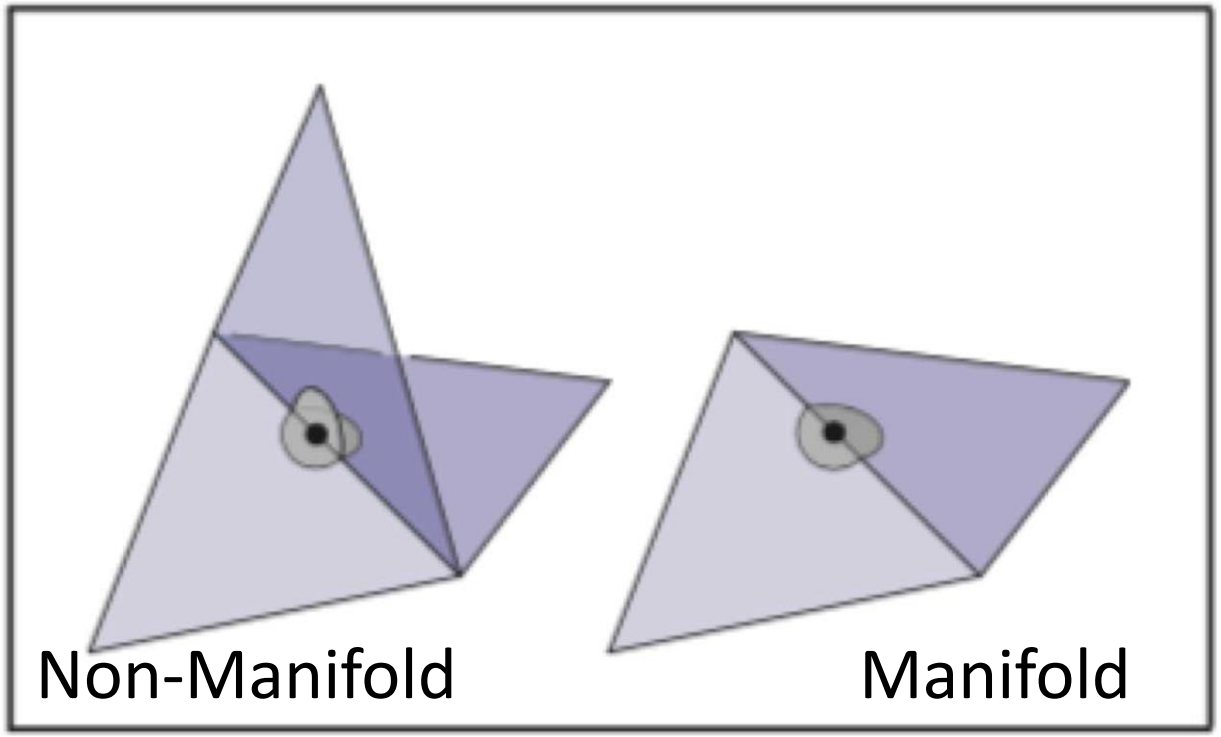


Mesh

Topology

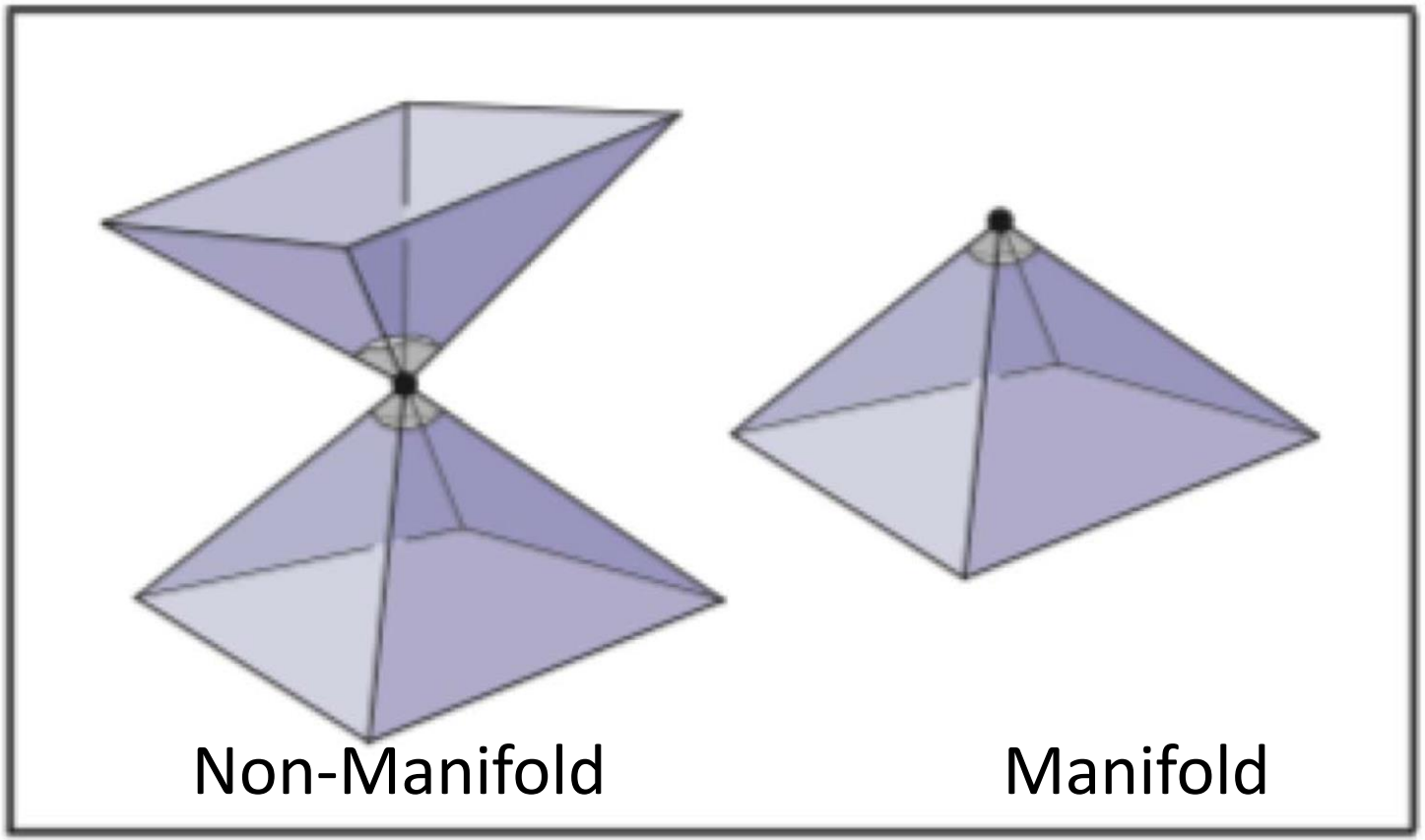
Topology is concerned with the connectivity of a mesh

We are going to assume that our meshes are *2-manifolds*



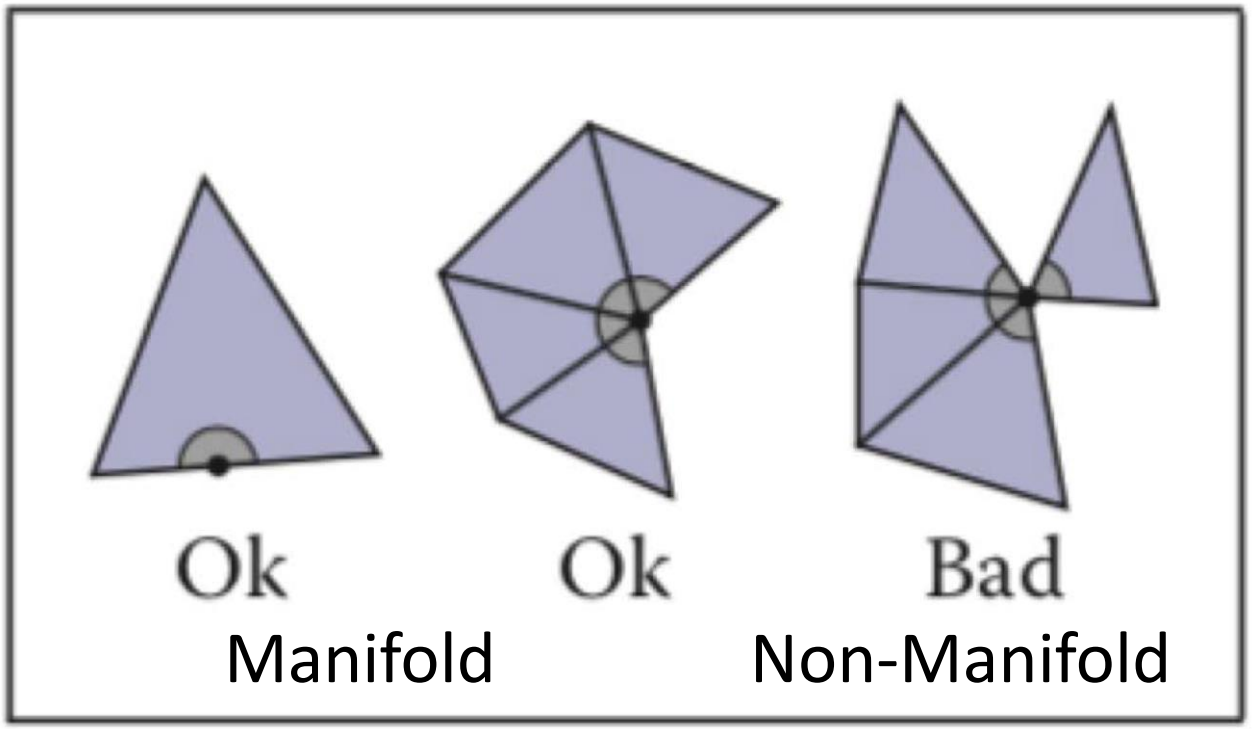
Manifold

A *2-manifold* is a surface for which the neighbourhood around any point can be flattened onto the plane



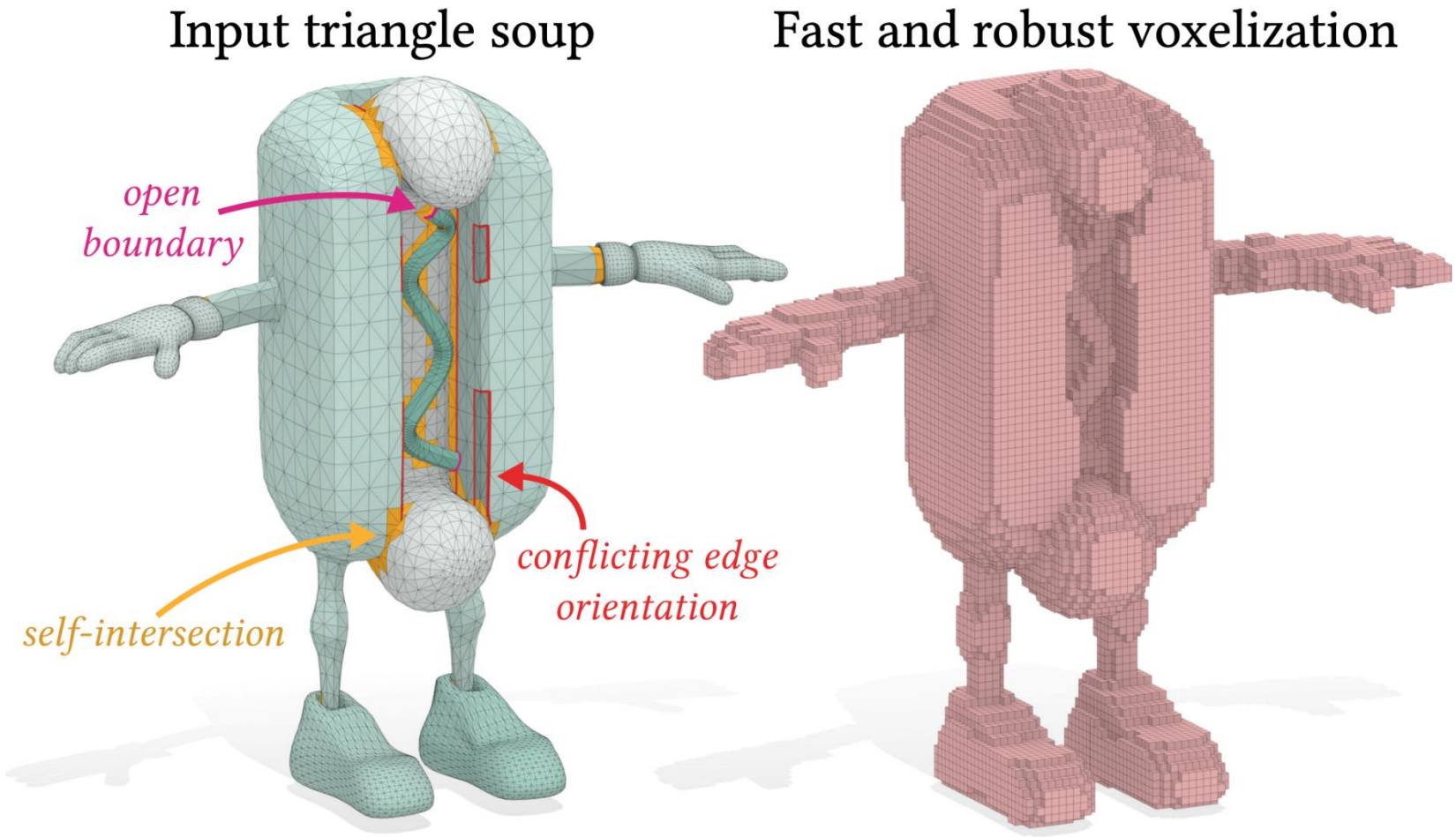
Manifold

A *2-manifold* is a surface for which the neighbourhood around any point can be flattened onto the plane



Watertight

Watertight meshes have no holes



https://www.dgp.toronto.edu/projects/fast-winding-numbers/

Geometry

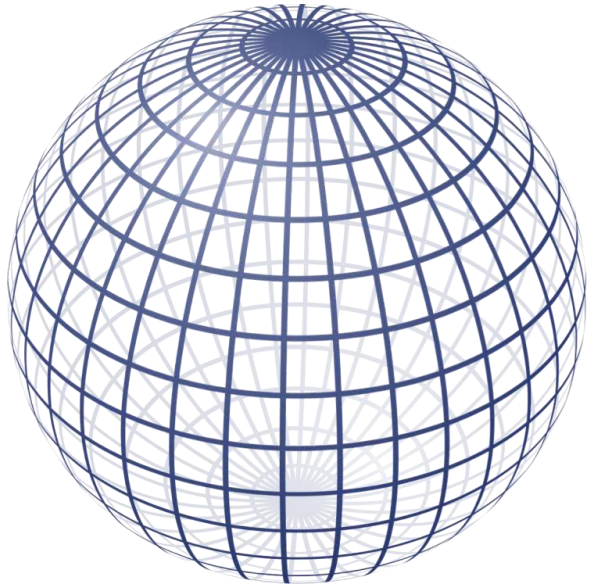
Geometrically, a mesh is a **piecewise planar** surface

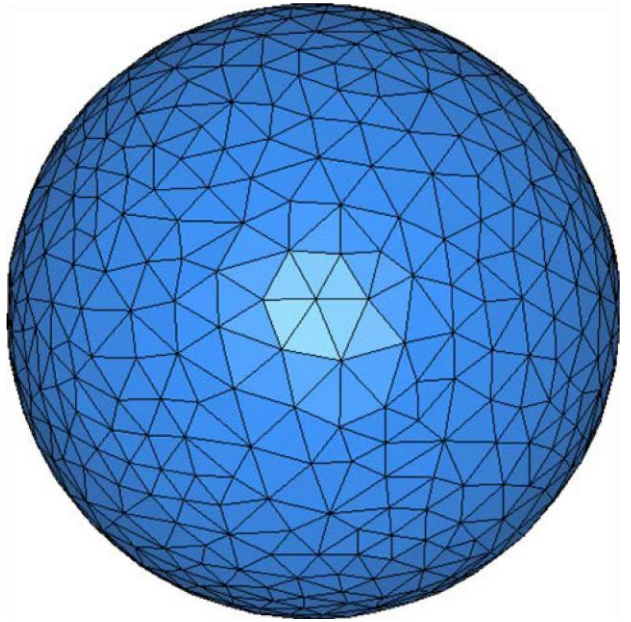
-almost everywhere, it is planar

-exceptions are at the edges where triangles join

Often, it's a piecewise planar approximation of a smooth surface

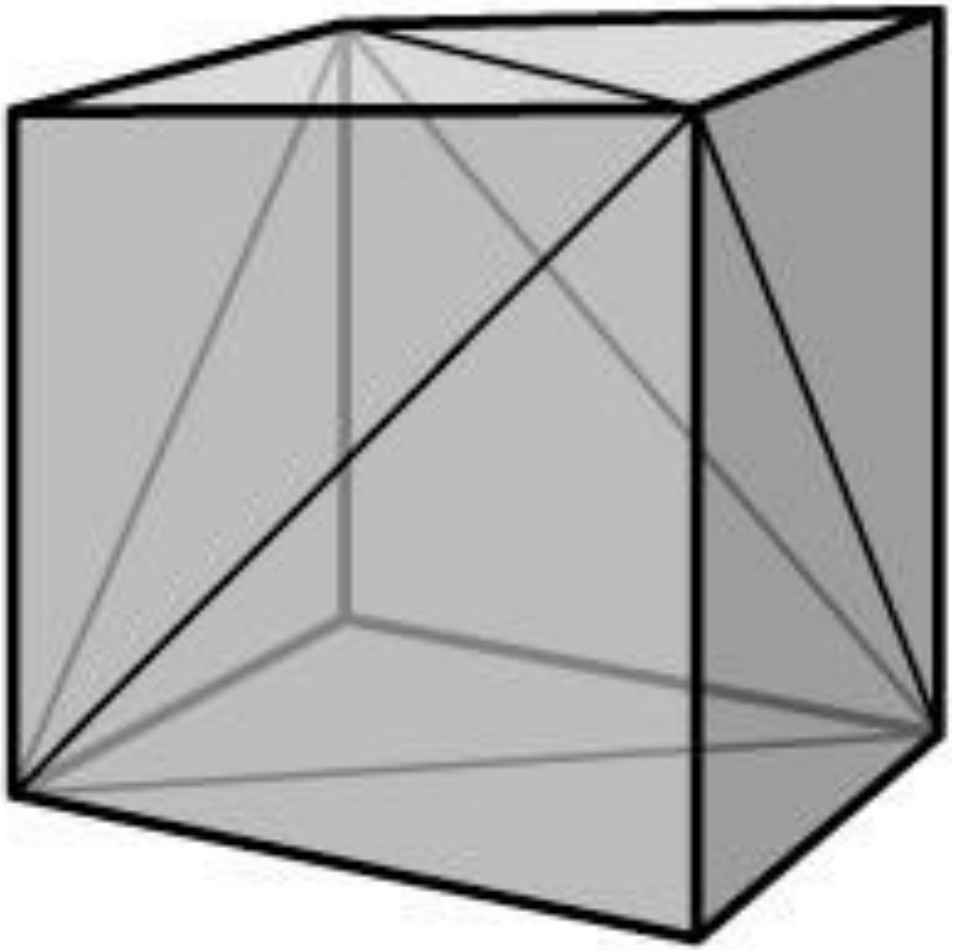
$$x = r\cos\phi\sin\theta,$$
$$y = r\sin\phi\sin\theta,$$
$$z = r\cos\theta.$$





Rineau & Yvinec CGAL manual

Examples of Meshes



12 triangles, 8 vertices

Examples of Meshes

10 million triangles from a high-resolution 3D scan

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


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

Storing Triangle Meshes

What do we care about ?

Storing Triangle Meshes

What do we care about?

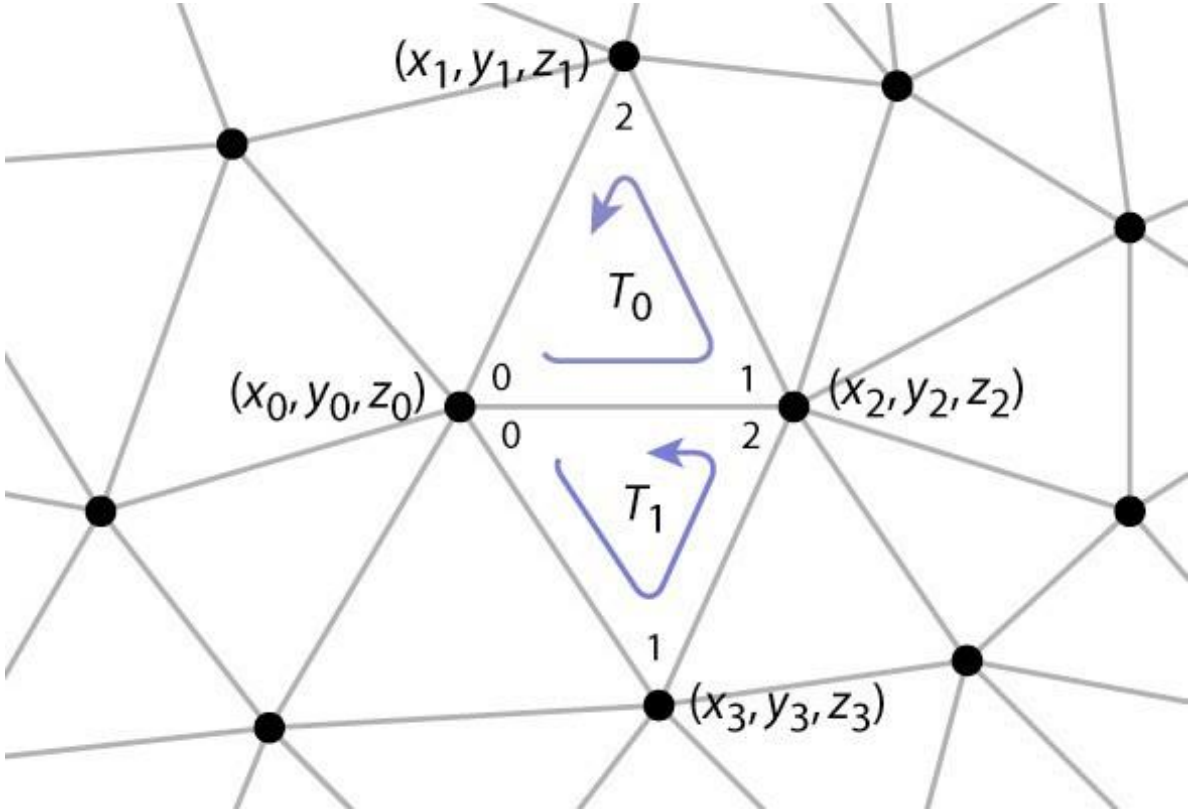
1. Compactness
2. Efficiency of queries
    - all vertices of a triangle
    - all triangles around a vertex
    - neighboring triangles of a triangle

Data Structures for Triangle Meshes

- Separate Triangles (soup)
- Indexed Triangle Set
- Triangle-Neighbour Data Structure
- Winged-Edge Data Structure
- Half-Edge Data Structure

Separate triangles

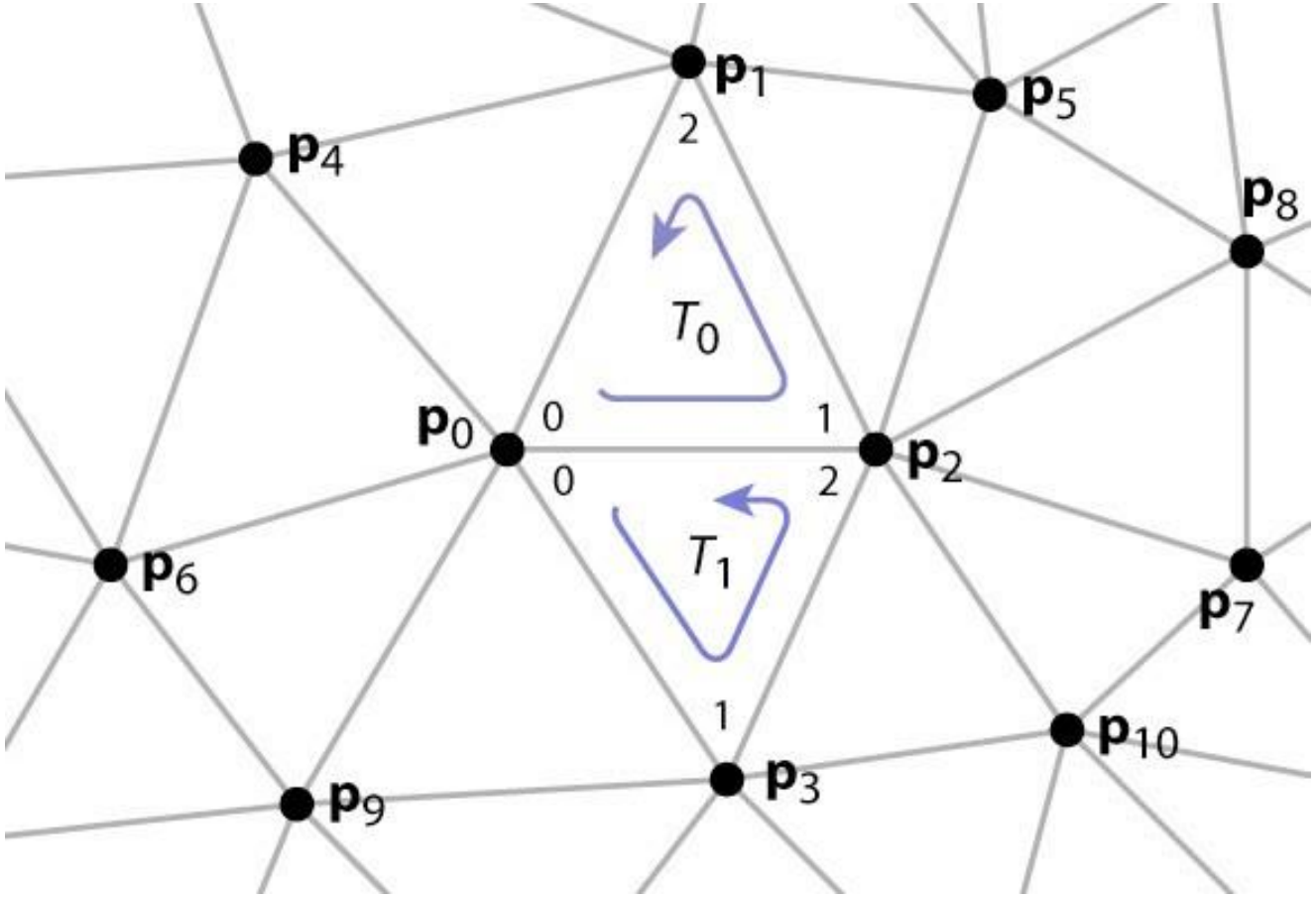
	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮



Indexed triangle set

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

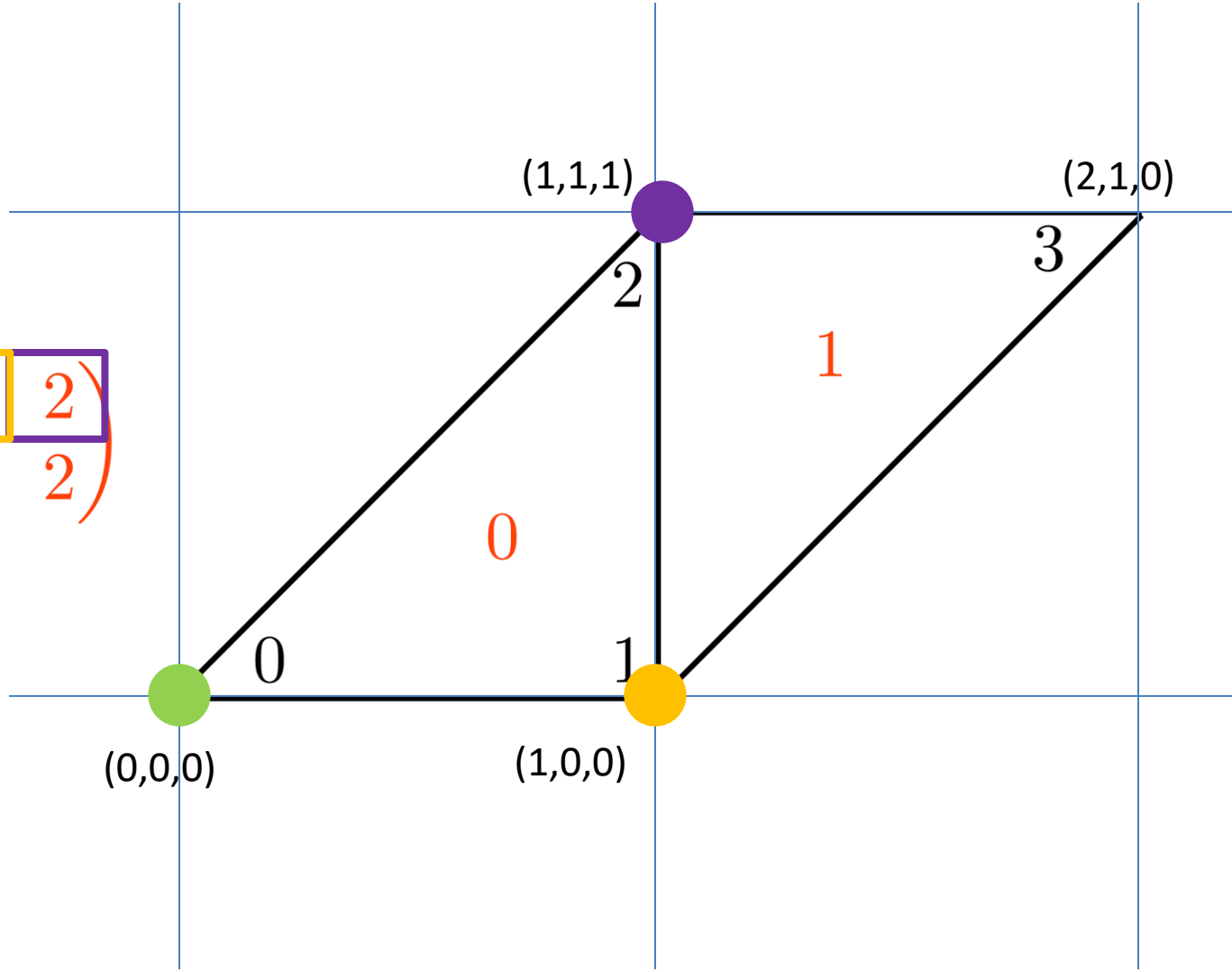
tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮



Indexed Triangle set

$$V = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

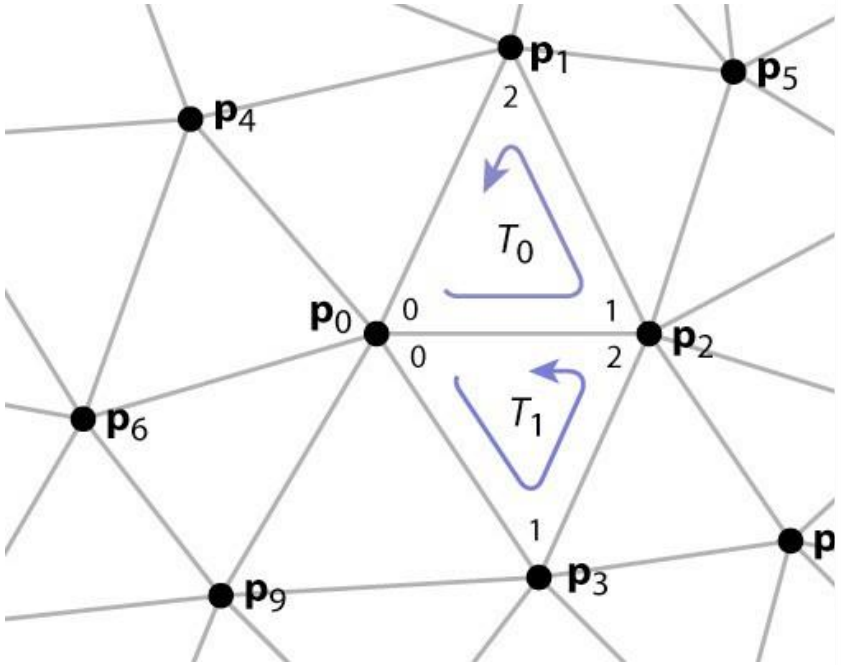
$$F = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 3 & 2 \end{pmatrix}$$

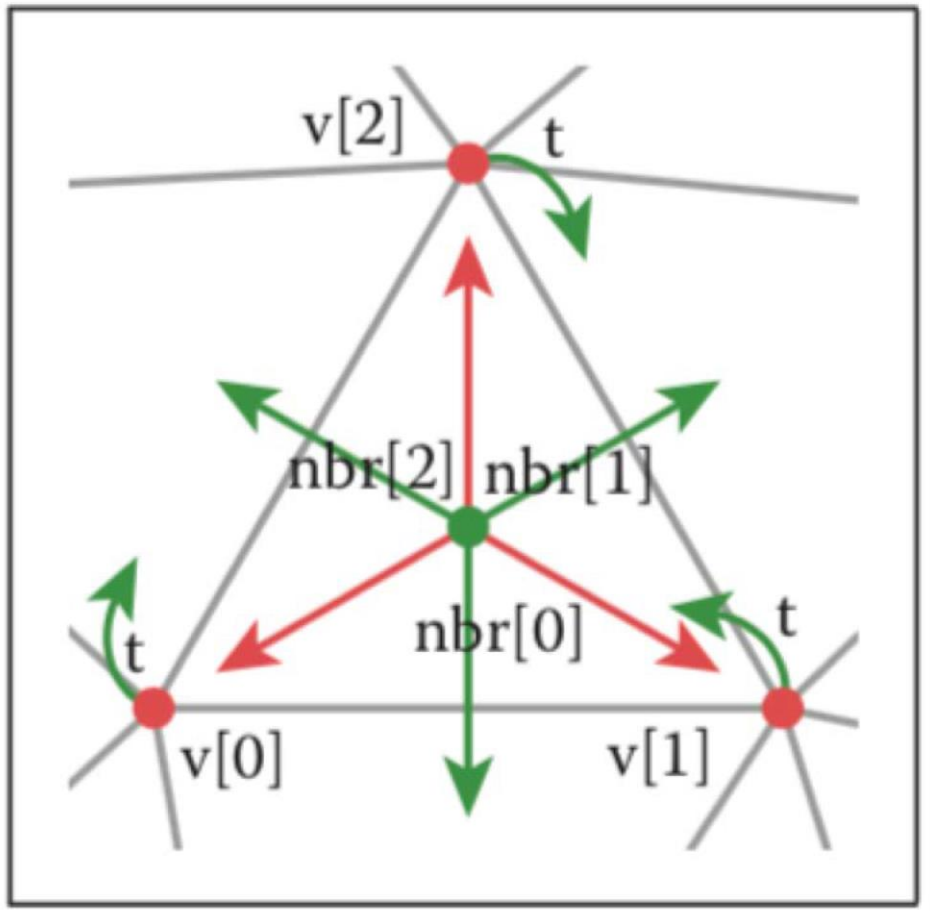


Triangle-Neighbour Data Structure

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

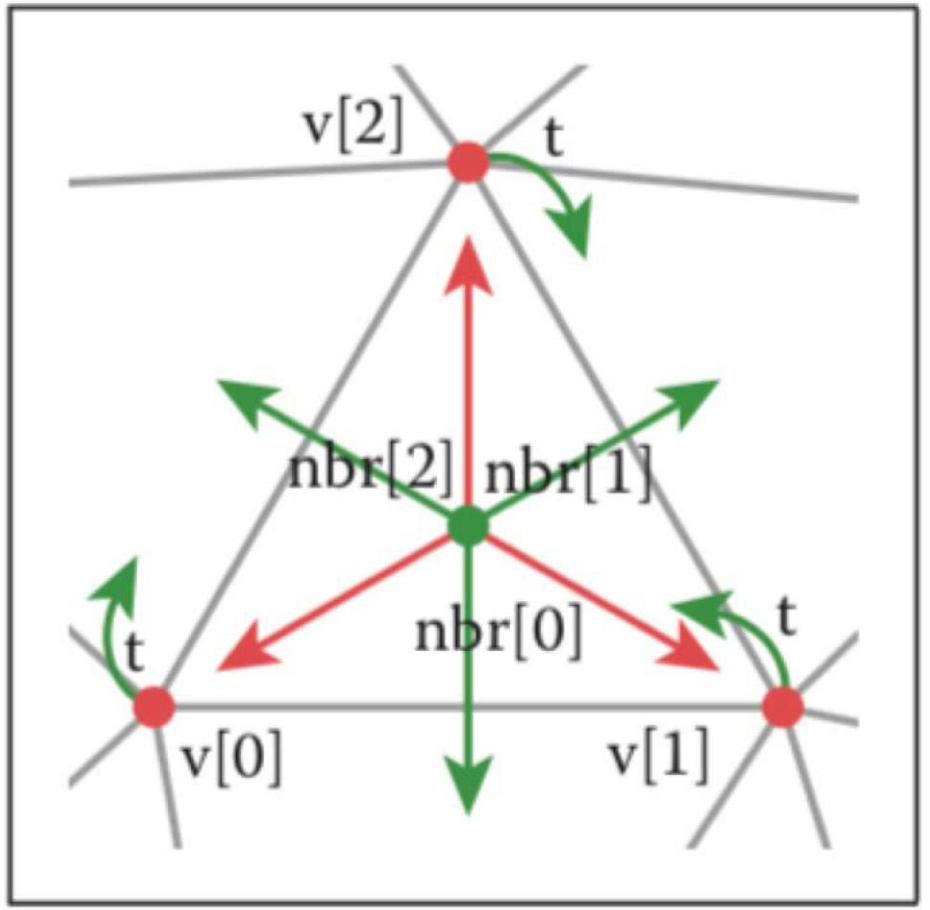
tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮





Triangle-Neighbour Data Structure

```
Mesh {
        float3 verts[nv];     // 3D vertex positions
        int vTri[nv];         // index of any adjacent triangle

        int tInd[nt][3];      // vertex indices
        int tNbr[nt][3];      // indices of neighbor triangles
        }
```

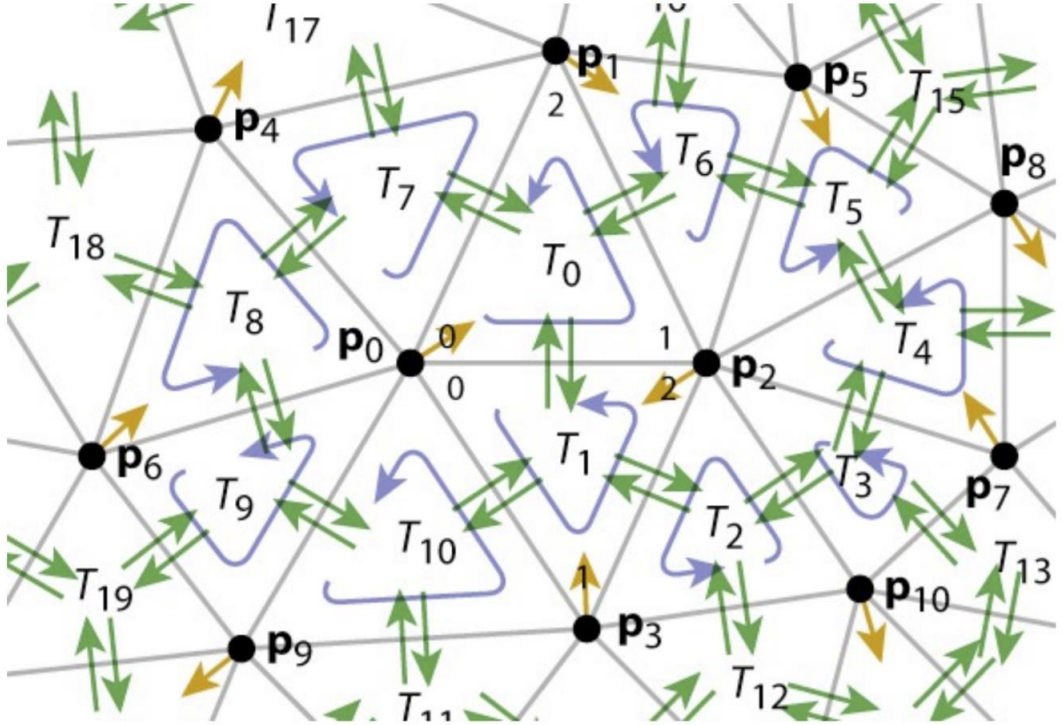


Triangle-Neighbour Data Structure

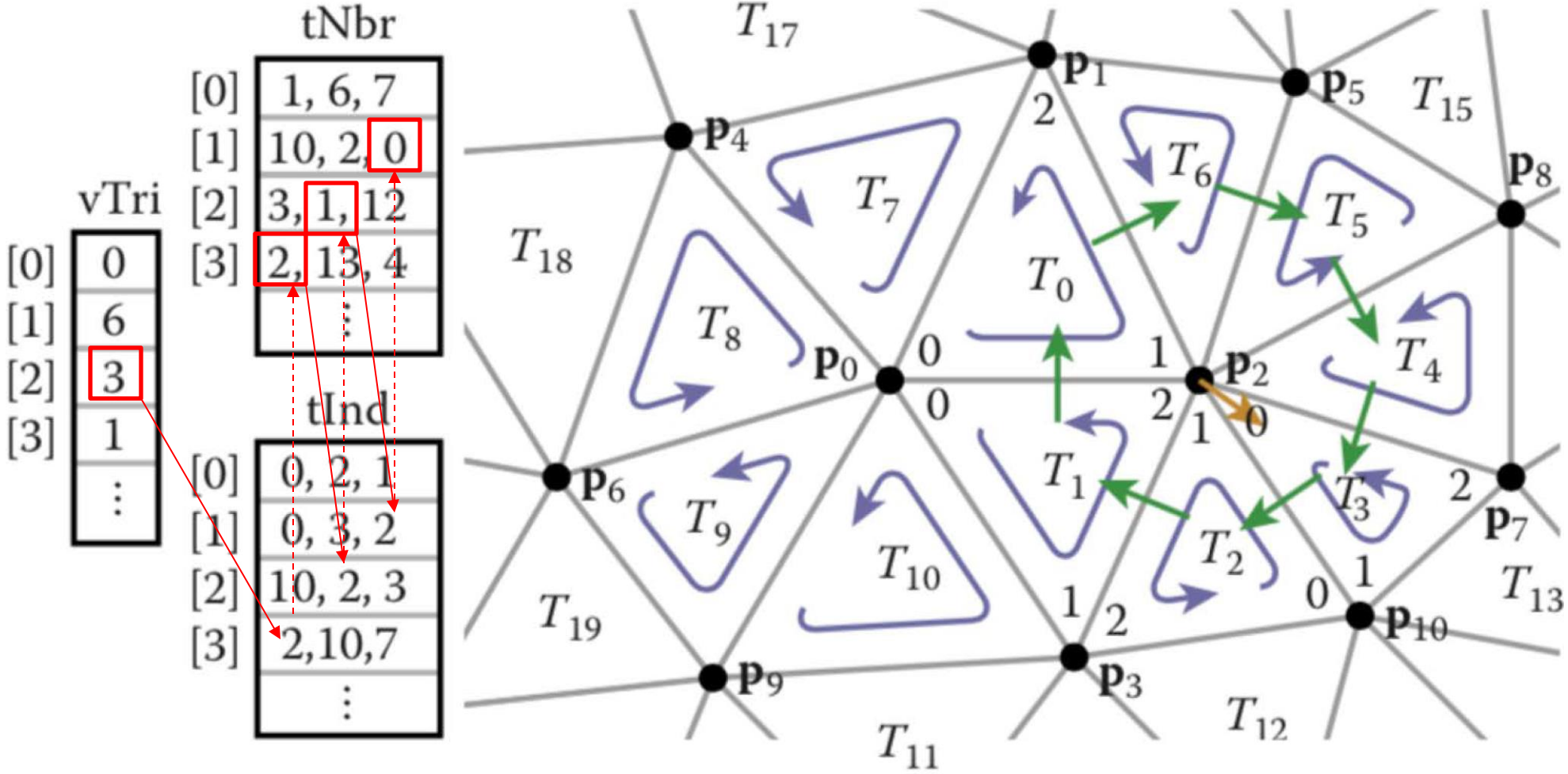
vTri[0]	0
vTri[1]	6
vTri[2]	1
vTri[3]	1
	⋮

tNbr[0]	1, 6, 7
tNbr[1]	10, 2, 0
tNbr[2]	3, 1, 12
tNbr[3]	2, 13, 4
	⋮

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
tInd[2]	10, 2, 3
tInd[3]	2, 10, 7
	⋮

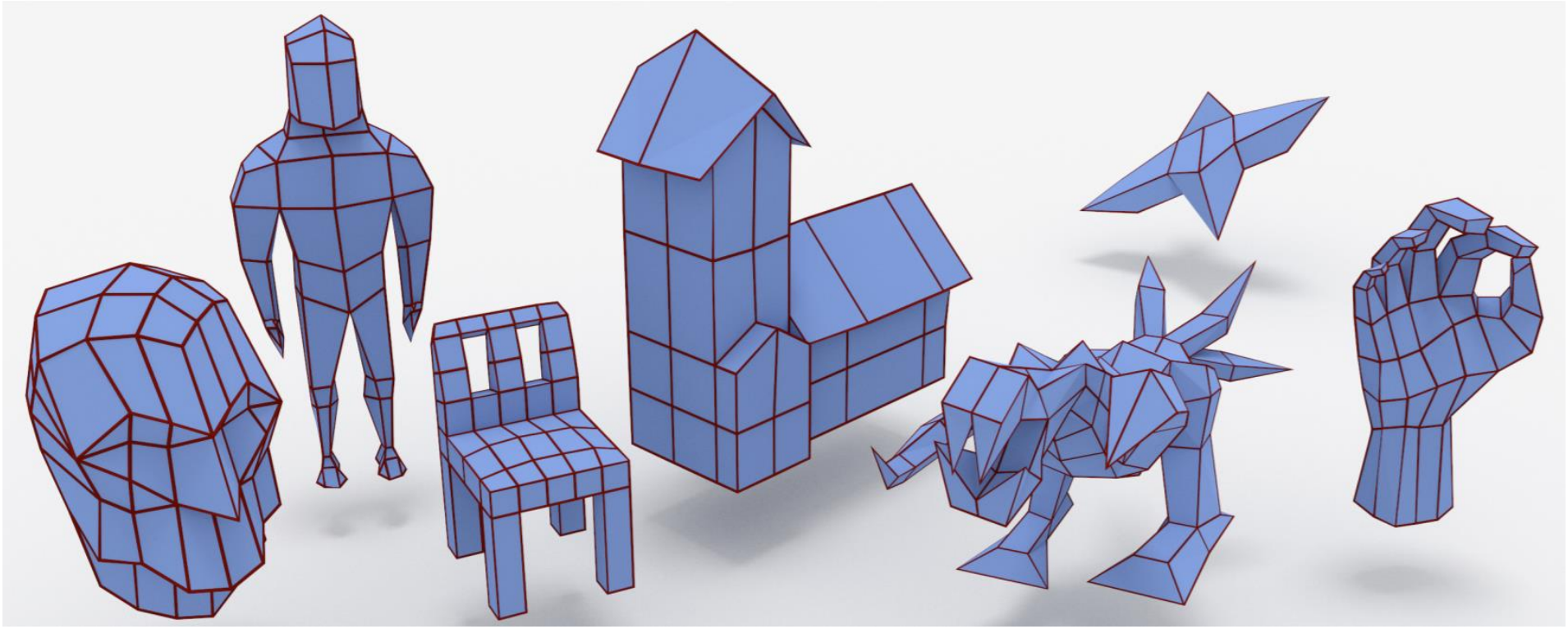


Triangle-Neighbour Data Structure



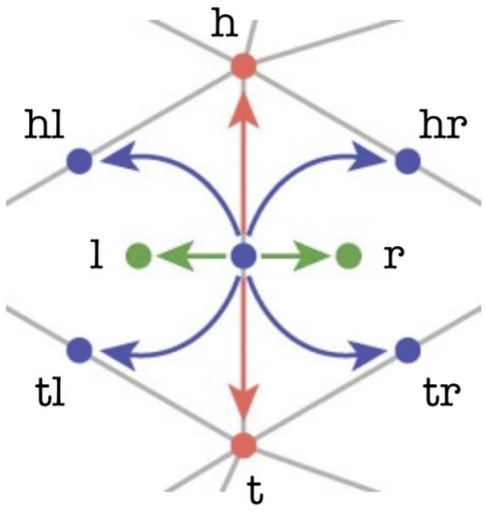
The kth entry of tNbr points to the neighboring triangle that shares vertices k and k+1

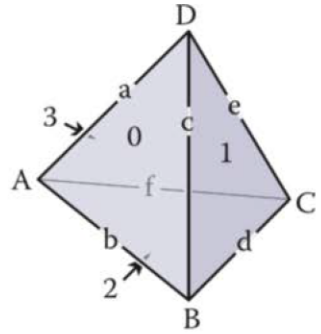
Quadrilateral (Quad) Meshes



[A. Bærentzen, J. Frisvad, and K. Singh. 2019. Signifier-Based Immersive and Interactive 3D Modeling. ACM VRST ‘19].

Winged-Edge Data Structure



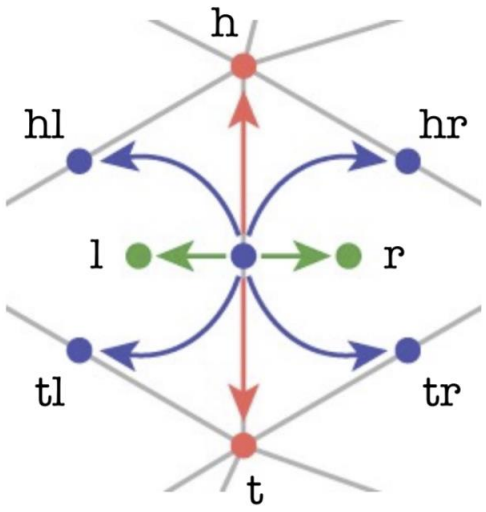


Edge	*Vertex 1*	*Vertex 2*	*Face left*	*Face right*	*Pred left*	*Succ left*	*Pred right*	*Succ right*
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	d	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	e	b	d

Vertex	*Edge*
A	a
B	d
C	d
D	e

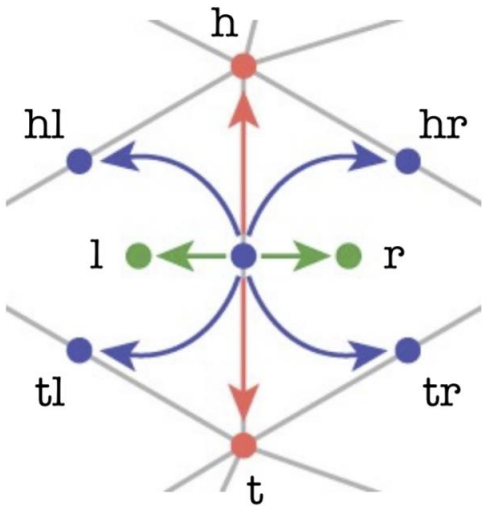
Face	*Edge*
0	a
1	c
2	d
3	a

Winged-Edge Data Structure



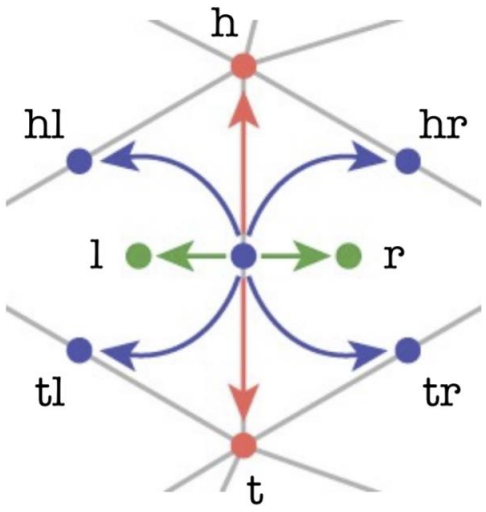

```
Edge {
        Edge hl, hr, tl, tr;
        Vertex h, t;
        Face l, r;
        }

Face { // per-face data
        Edge e; // any adjacent edge
        }

Vertex { // per-vertex data
        Edge e; // any incident edge
        }
```

Winged-Edge Data Structure



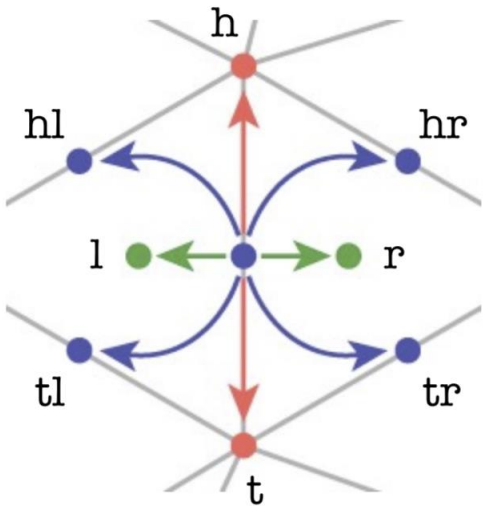

```
EdgesOfFace(f)
{
    e = f.e;
    do {
        if (e.l == f) e = e.hl;
        else e = e.tr;
    } while (e != f.e);
}
```

Winged-Edge Data Structure



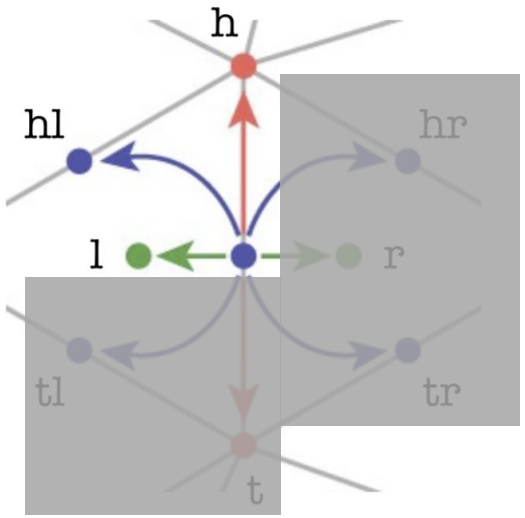

```
EdgesOfVertex(v)
{
    e = v.e;
    do {
          if (e.t == v) e = e.tl;
          else e = e.hr;
         } while (e != v.e);
}
```

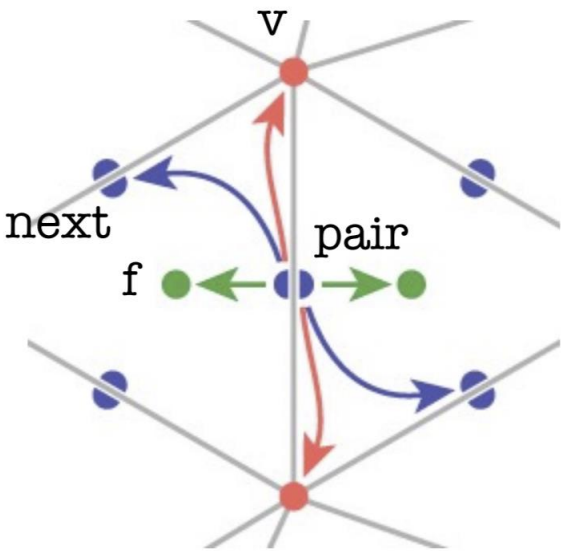
Winged-Edge Data Structure




```
EdgesOfVertex(v)
{
    e = v.e;
    do {
          if (e.t == v) e = e.tl;
          else e = e.hr;
         } while (e != v.e);
}

FacesOfVertex(v)
{
    e = v.e;
    do {
          if (e.t == v) {
             f=e.l;
             e = e.tl; }
          else {
             f=e.r;
             e = e.hr;}
         } while (e != v.e);
}
```

Half-Edge Data Structure






```
HEdge {
        Edge hl, hr, tl, tr;
        Vertex h, t;
        Face l, r;
        }
```

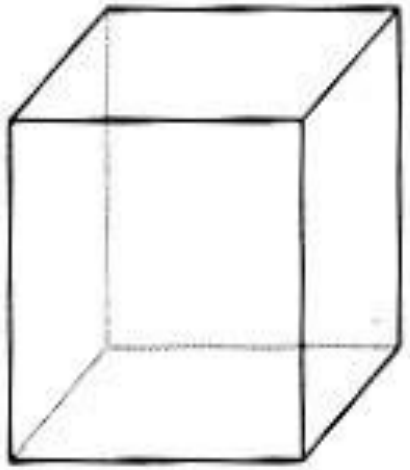
Hedge pair, next;
Vertex v;
Face f;

```
Face { // per-face data
        HEdge e; // any adjacent edge
        }

Vertex { // per-vertex data
        HEdge e; // any incident edge
        }

EdgesOfVertex(v) {
        h = v.h;
        do { h = h.next.pair; }
        while (h != v.h);
}
```

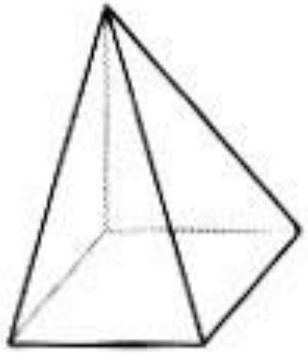
Relationships between primitive Types

What is the relationship between the number of vertices, the number of edges and the number of triangles in a mesh?

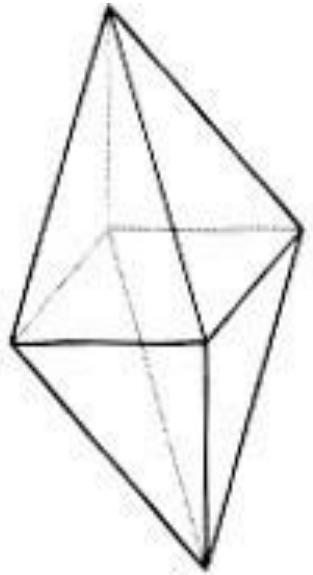
Euler's Formula: V+F-E = 2
(closed manifold mesh)



V = 8
E = 12
F = 6



V = 5
E = 8
F = 5



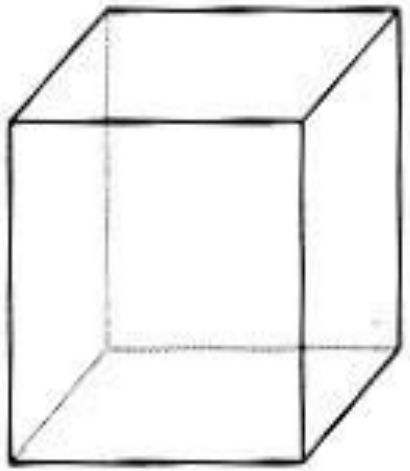
V = 6
E = 12
F = 8

Relationships between primitive Types

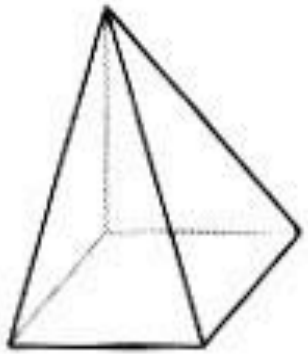
Number of half edges = 2E = 3F

(closed triangle mesh)

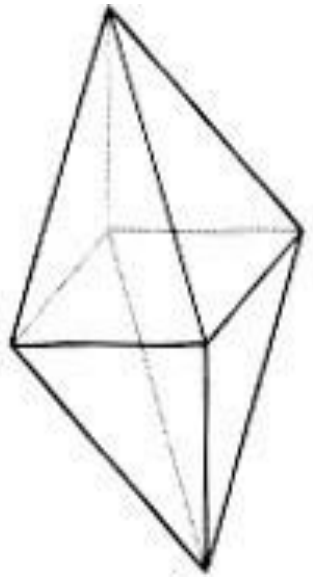
V+F-3F/2=2 => F=2V-4



V = 8
E = 12
F = 6



V = 5
E = 8
F = 5



V = 6
E = 12
F = 8

Data on meshes

Often need to store additional information besides just the geometry

Can store additional data at faces, vertices, or edges

Examples

- colours 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

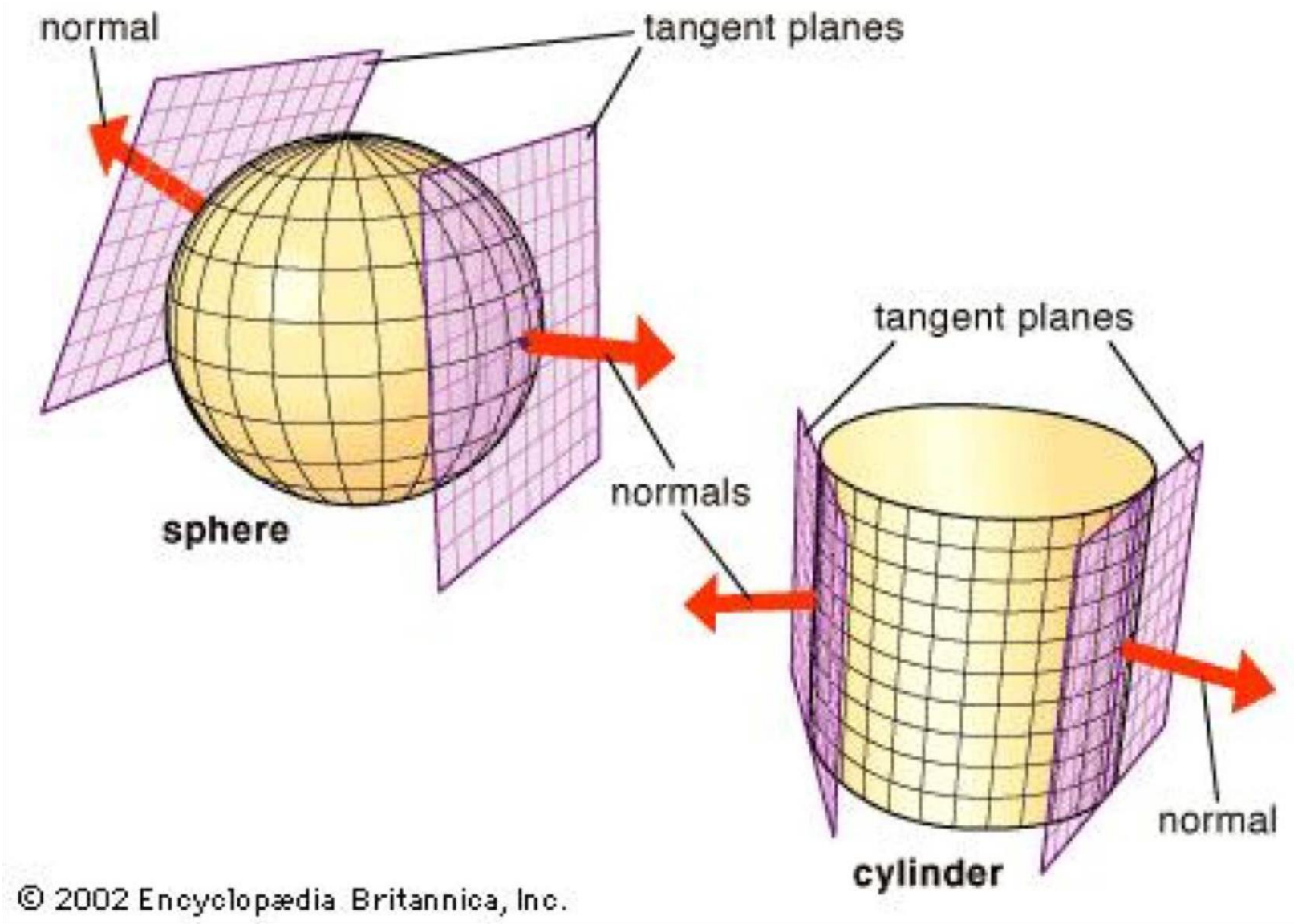
Key types of vertex data

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

- Surface normals
  - when a mesh is approximating a curved surface, store normals at vertices

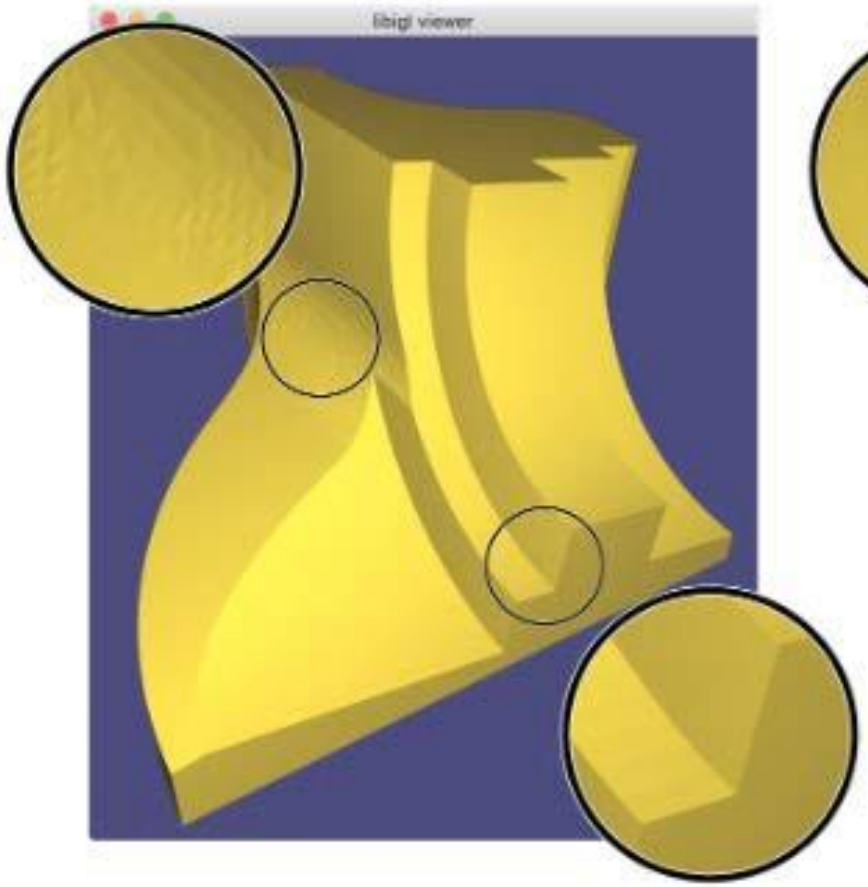
- Texture coordinates
  - 2D coordinates that tell you how to paste images or 2D data on the surface

Normal Vectors

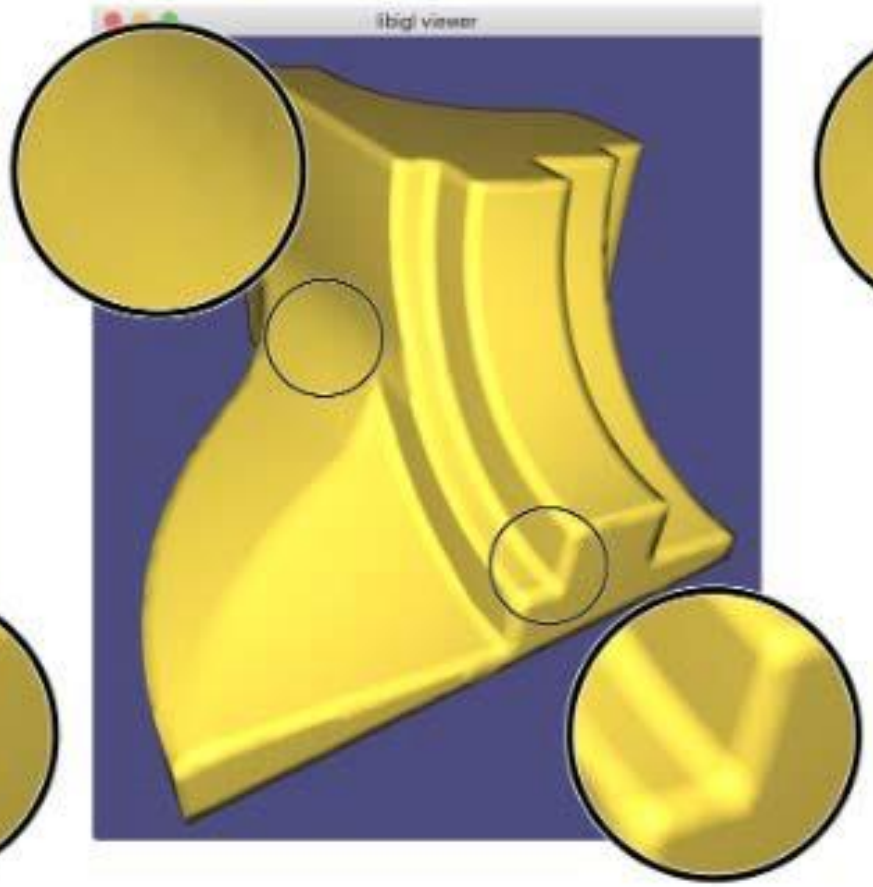


Computing Per-vertex Normals

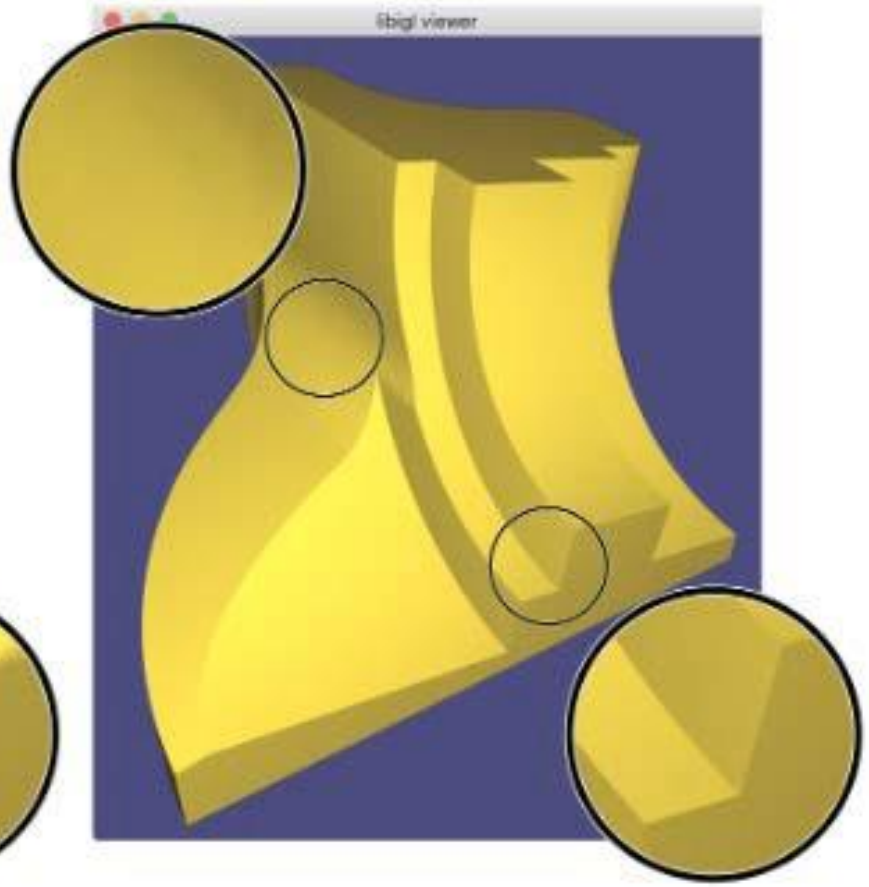
Per-Face Normals



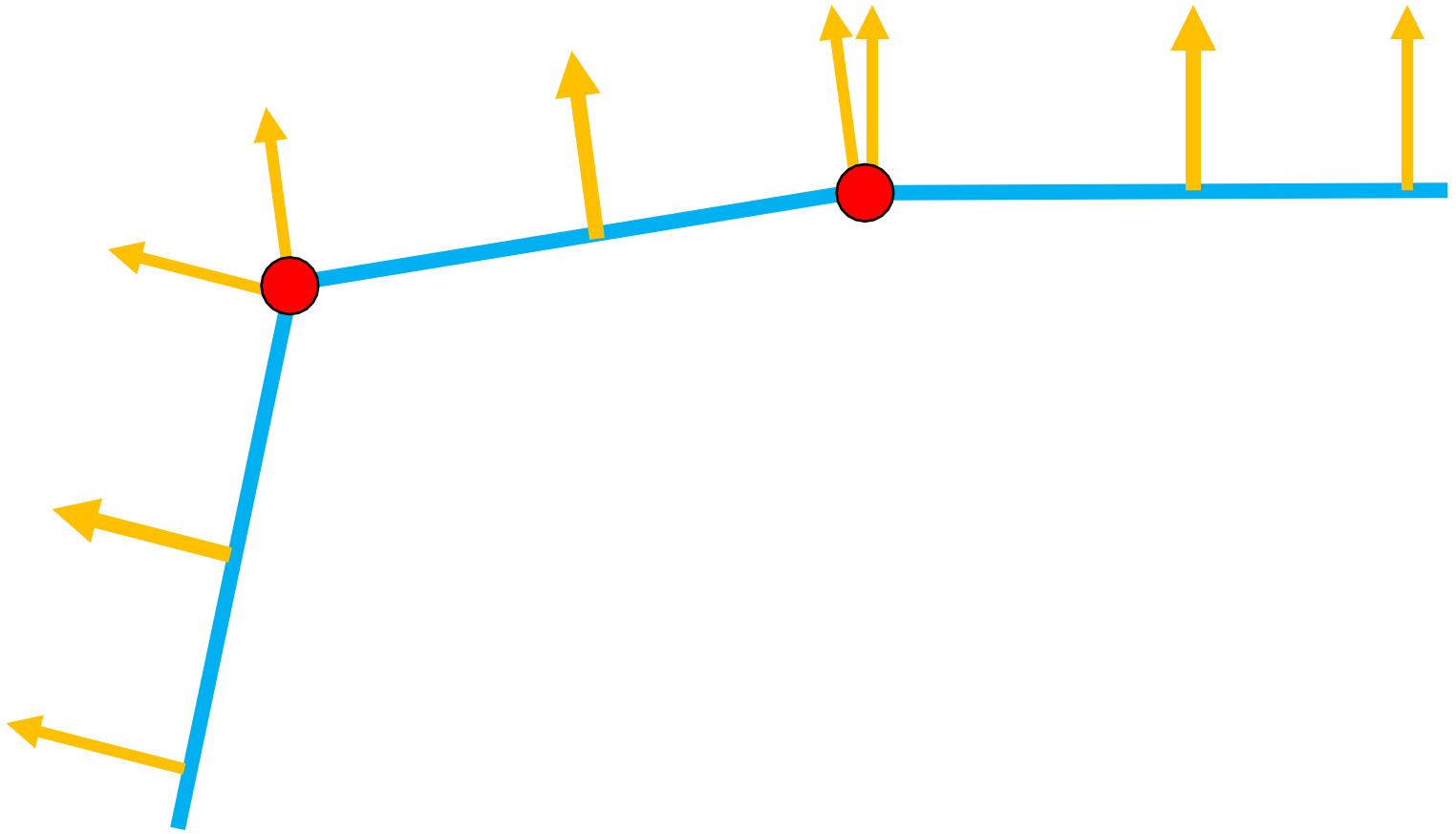
Per-Vertex Normals



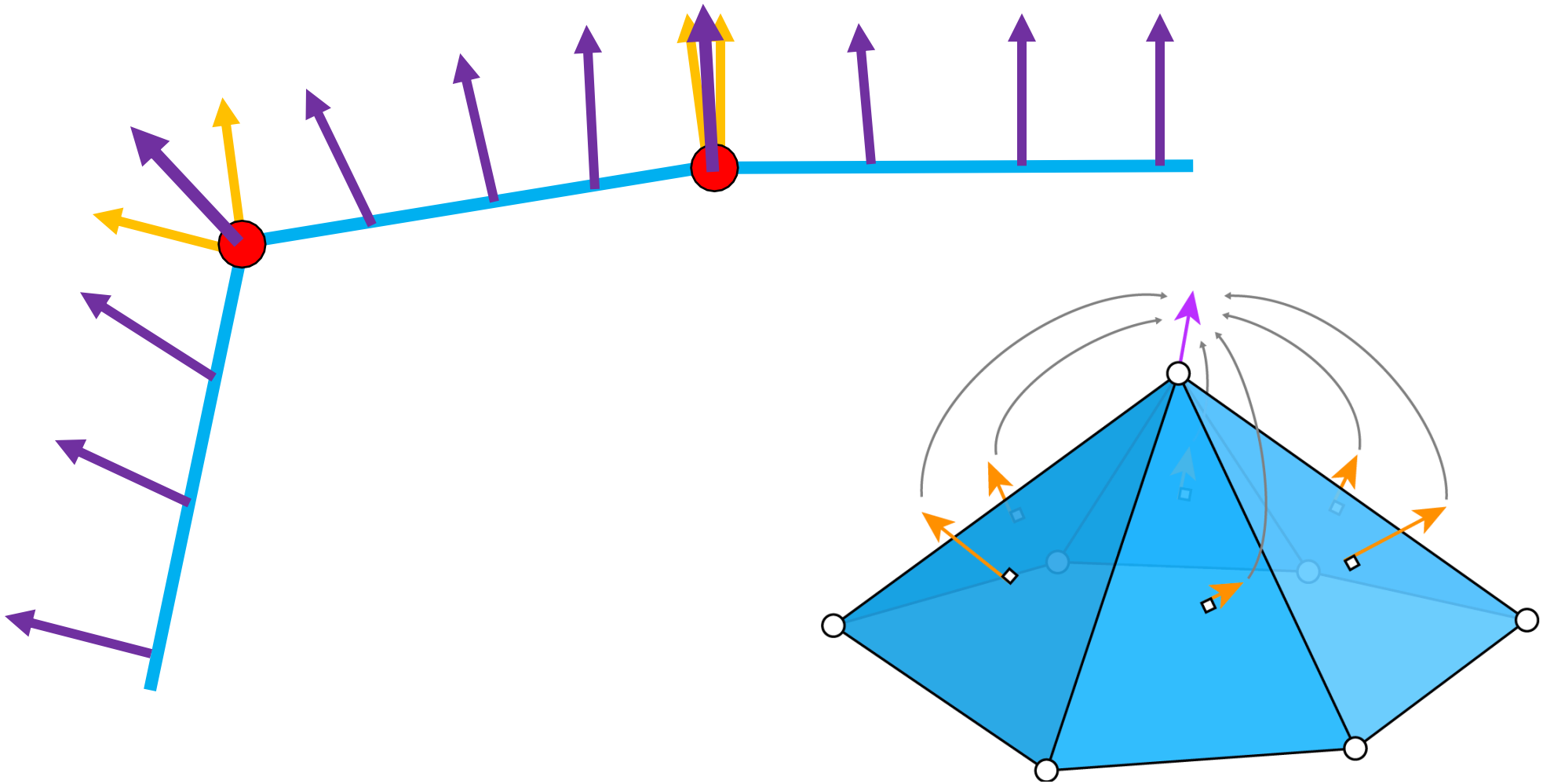
Per-Corner Normals



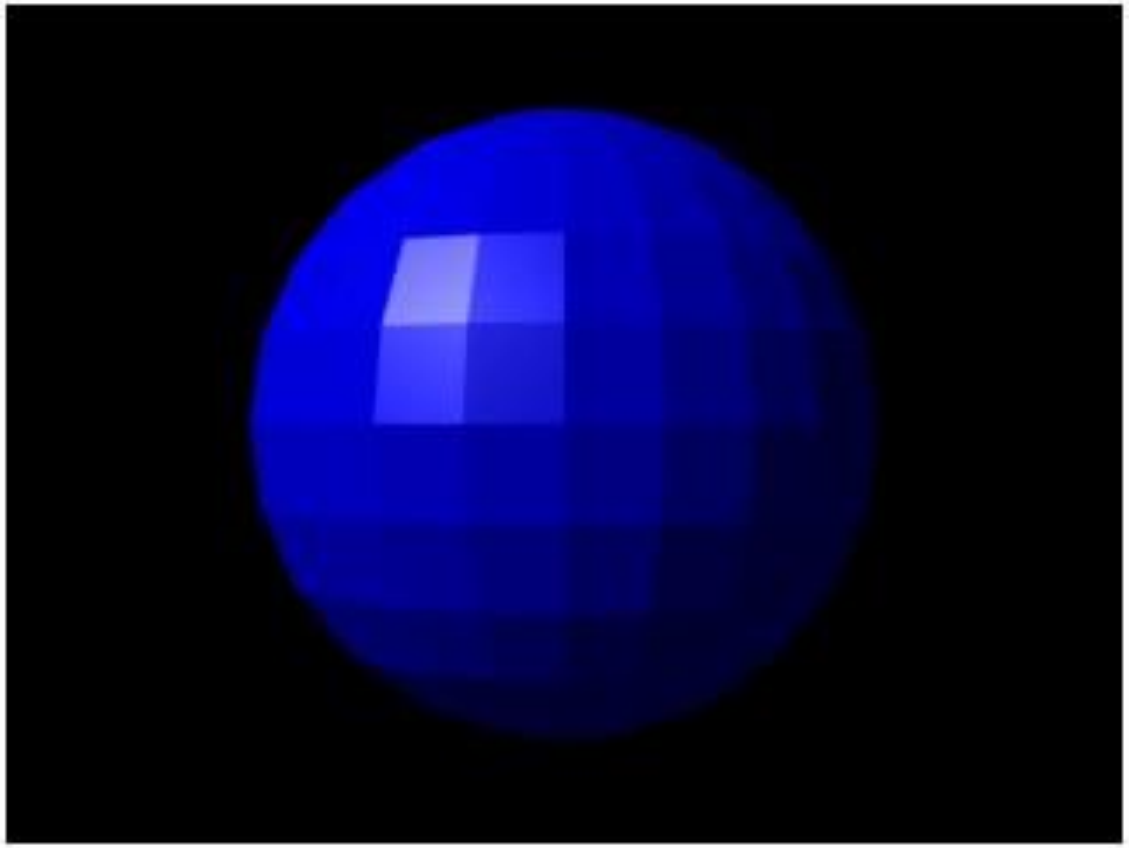
Per-Face Normals

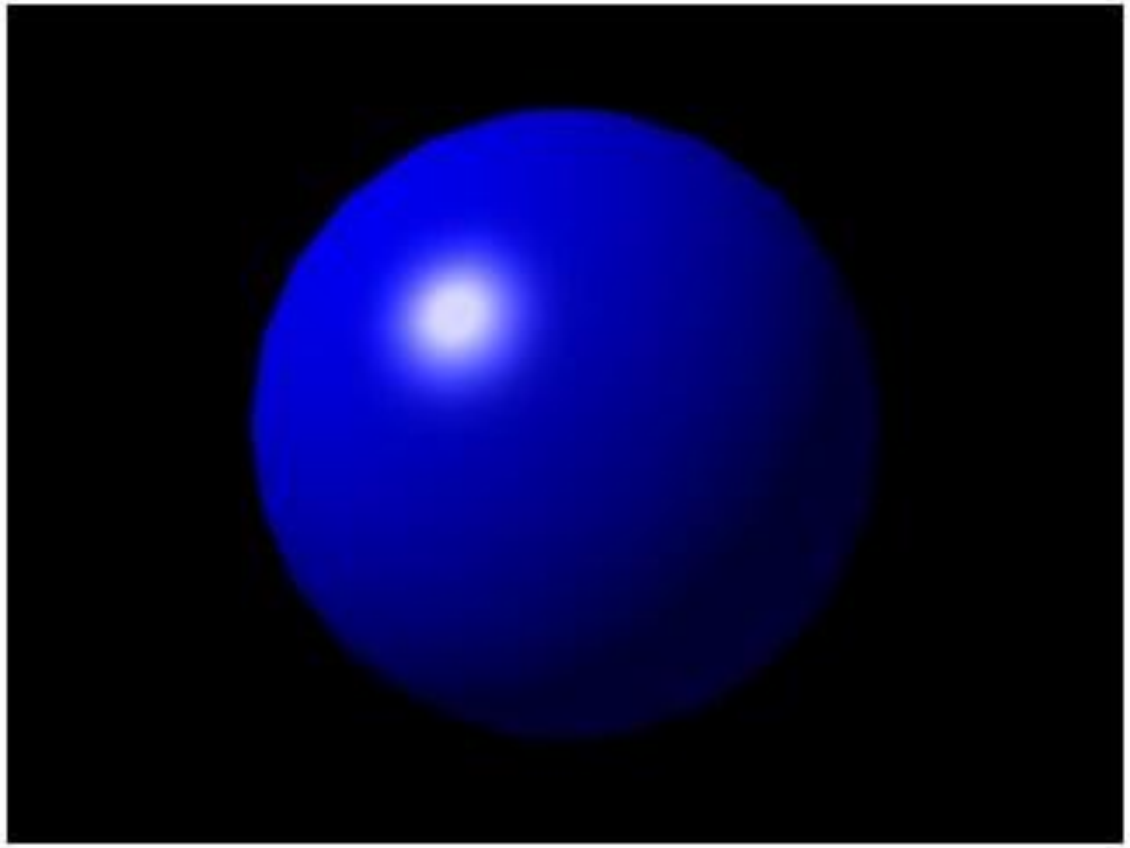


Per-Vertex Normals



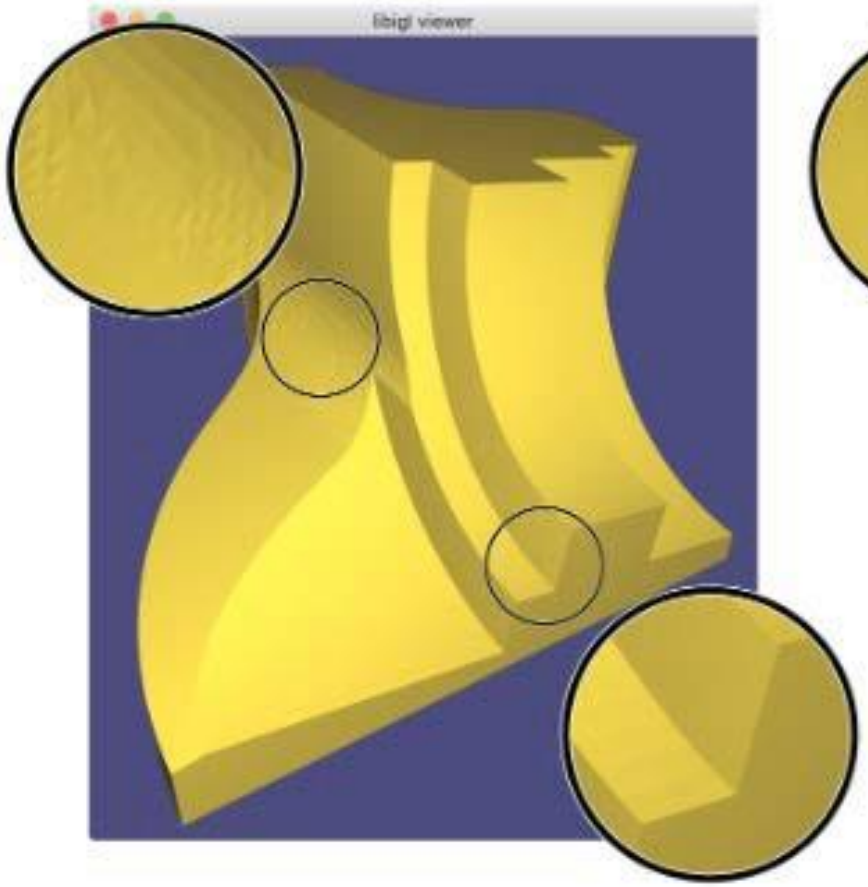
Meshes representing smooth surfaces



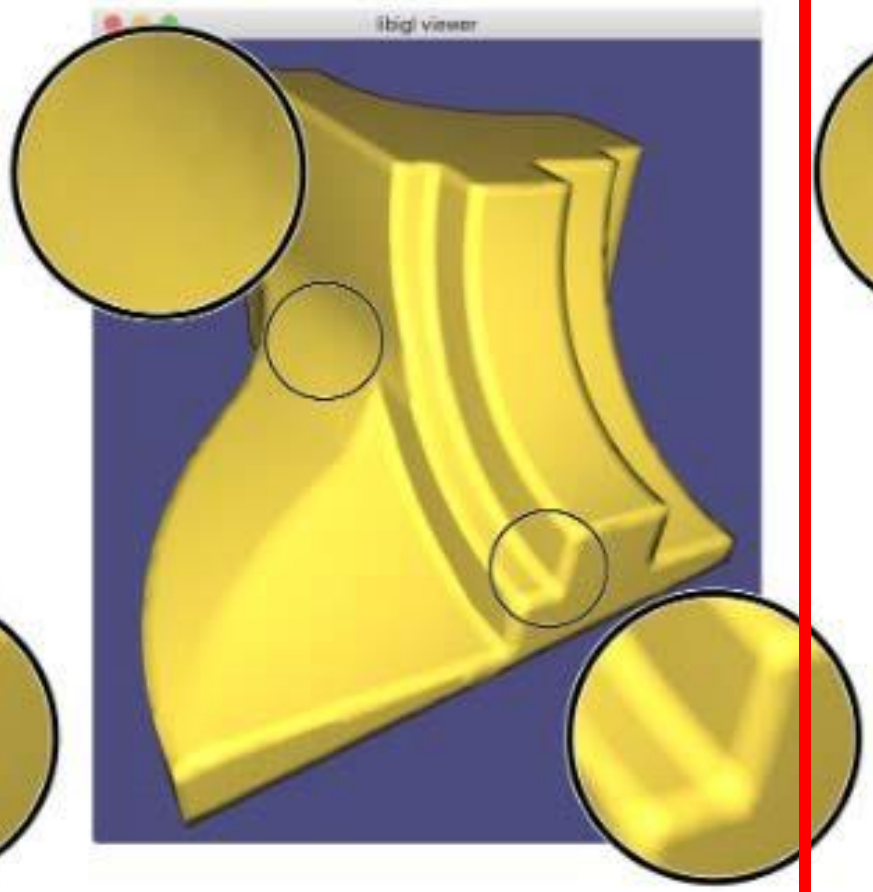


Per-Corner Normals

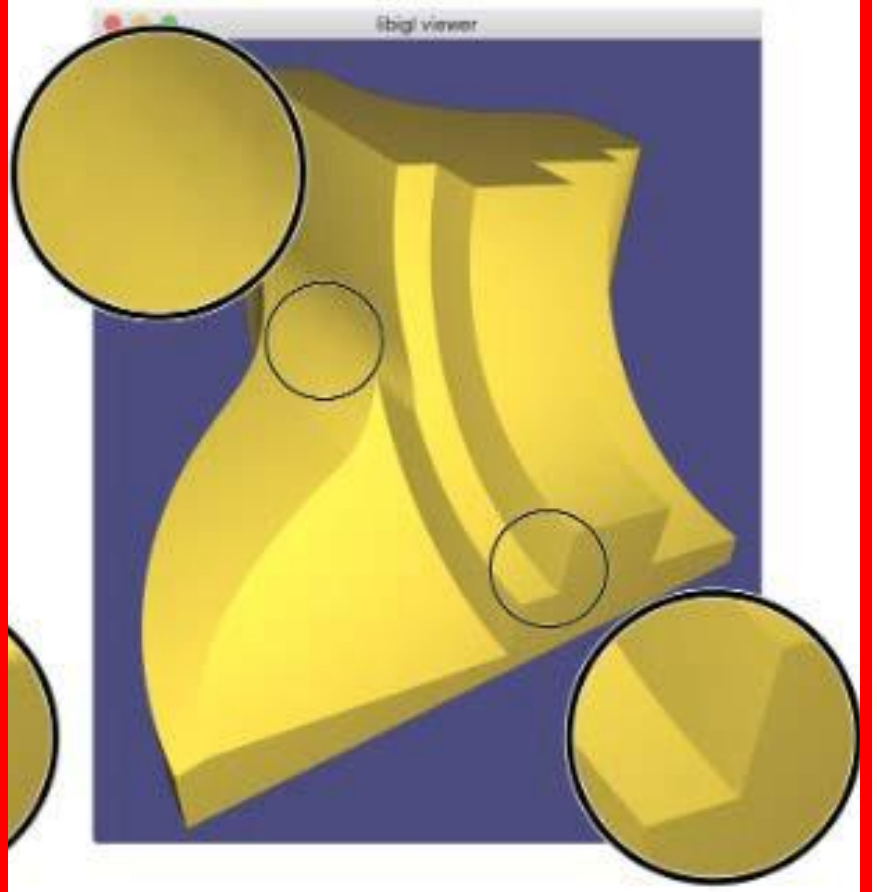
Per-Face Normals



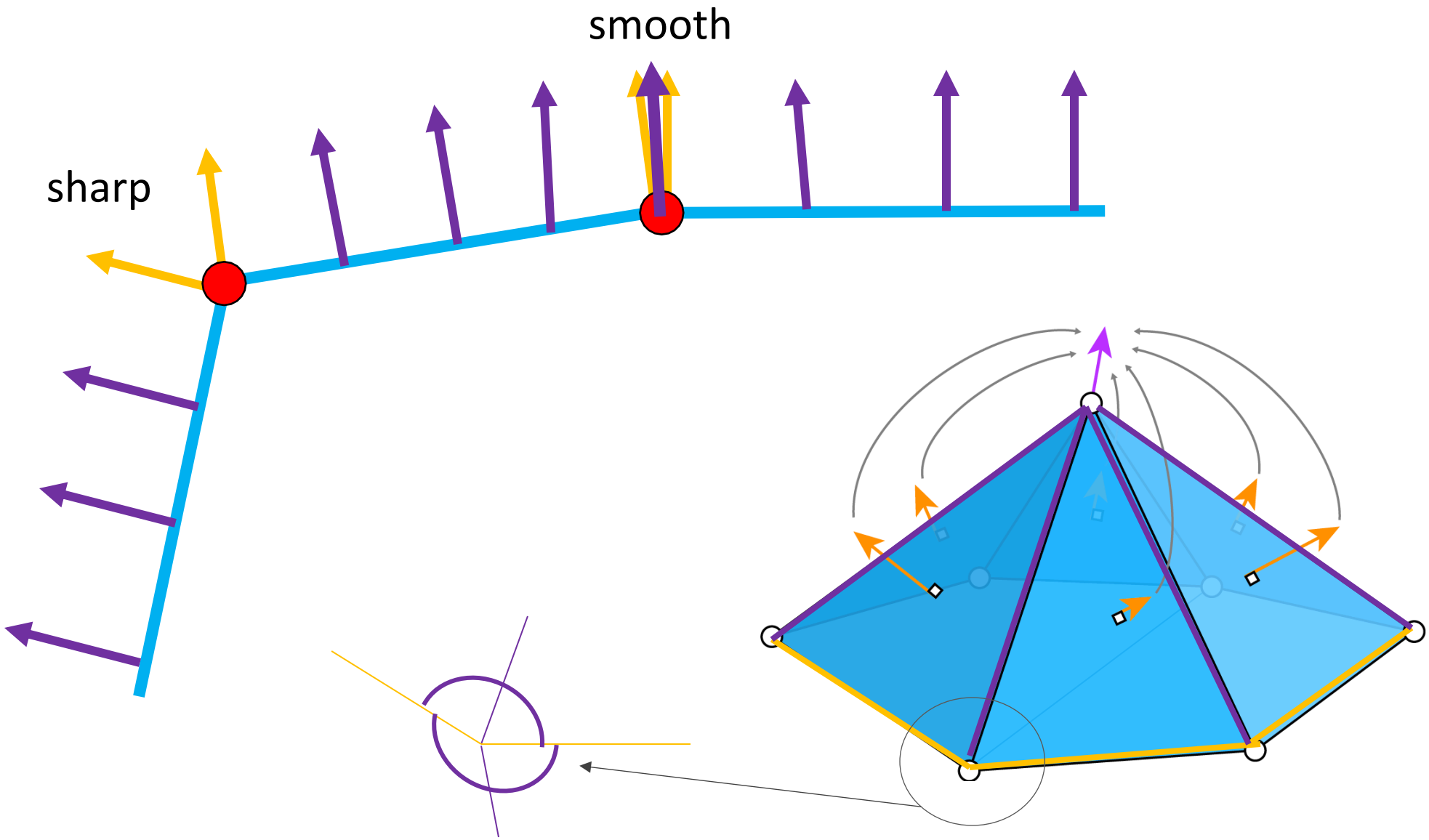
Per-Vertex Normals



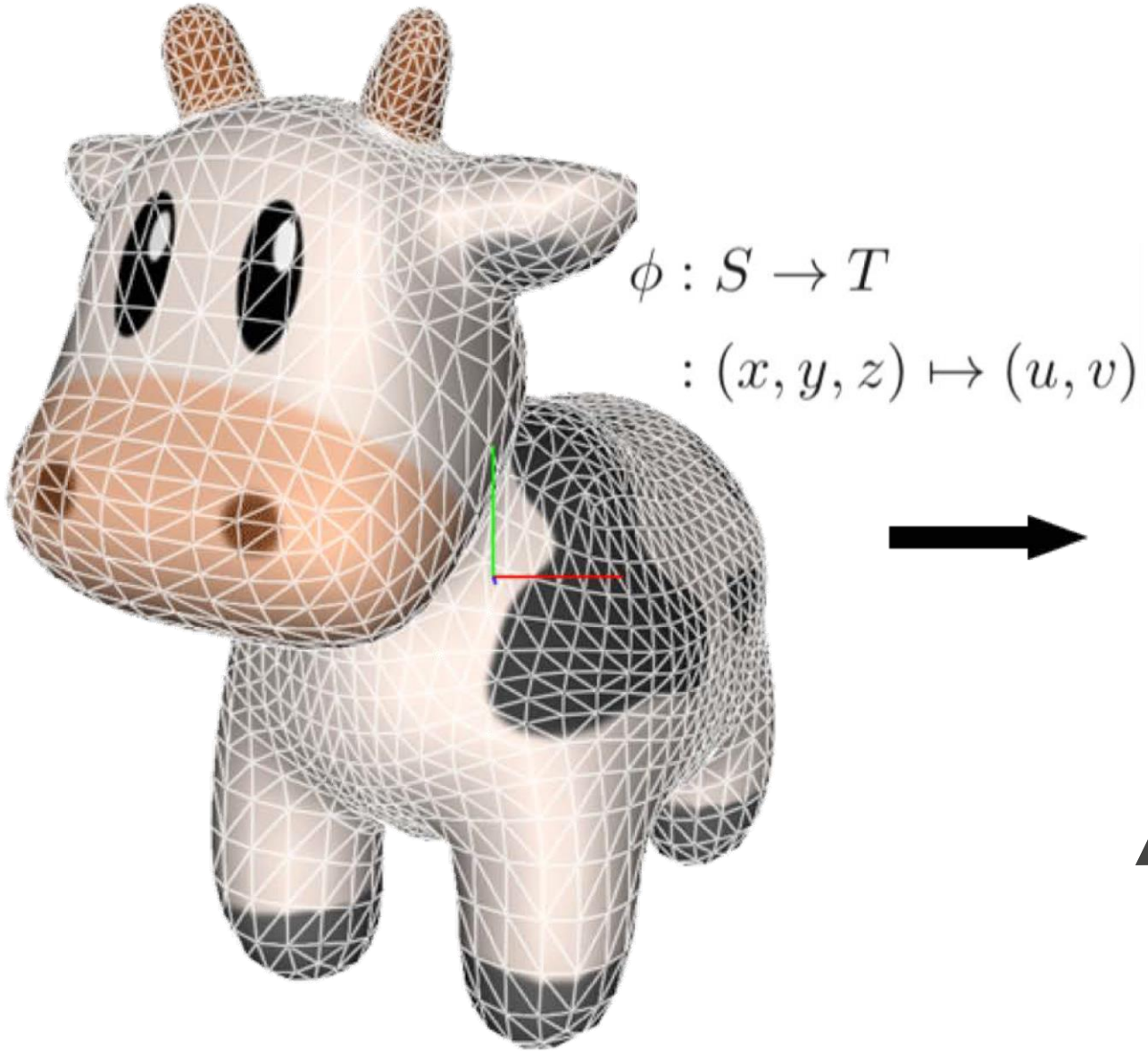
Per-Corner Normals



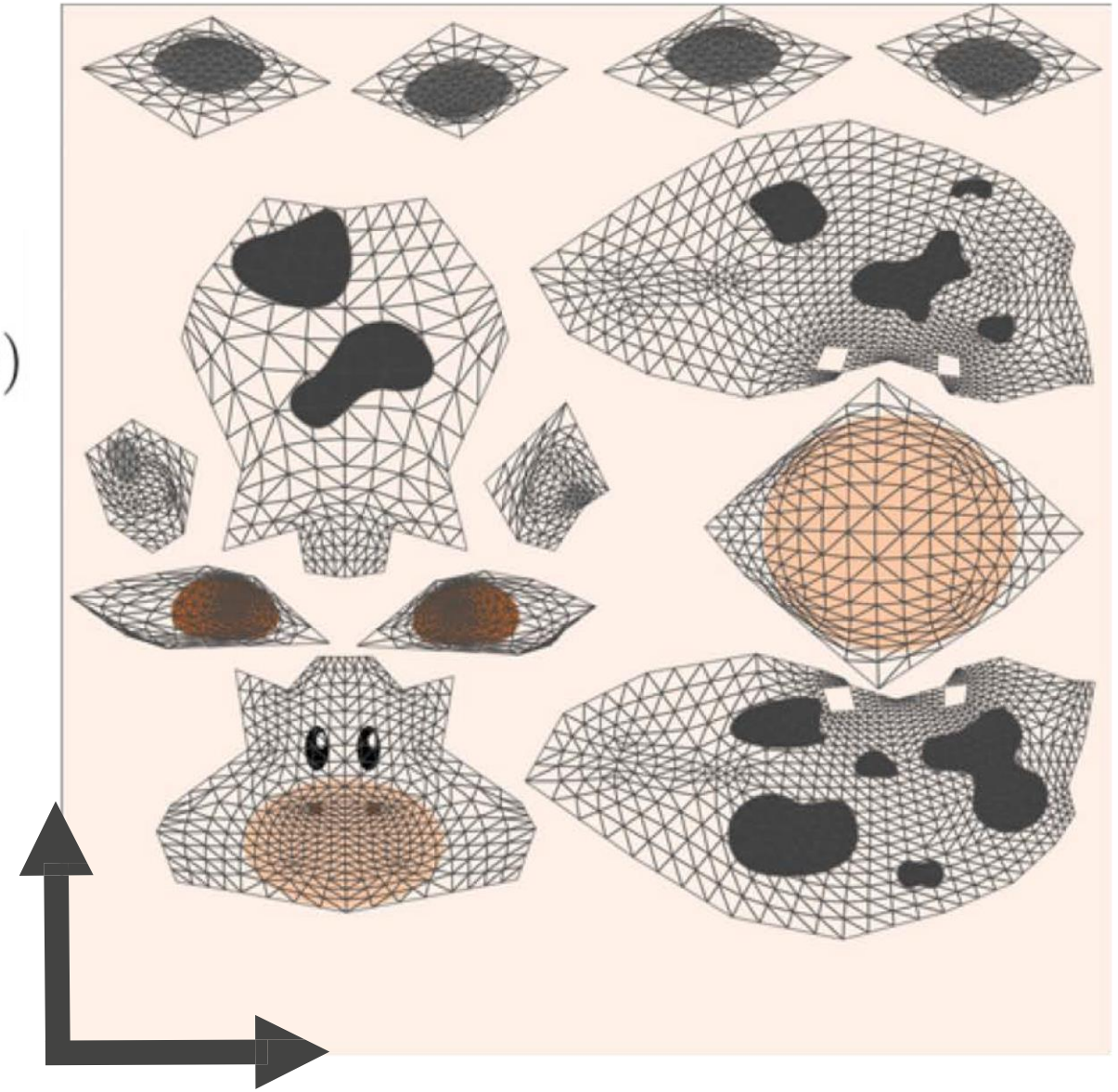
Per-Corner Normals



Texture coordinates



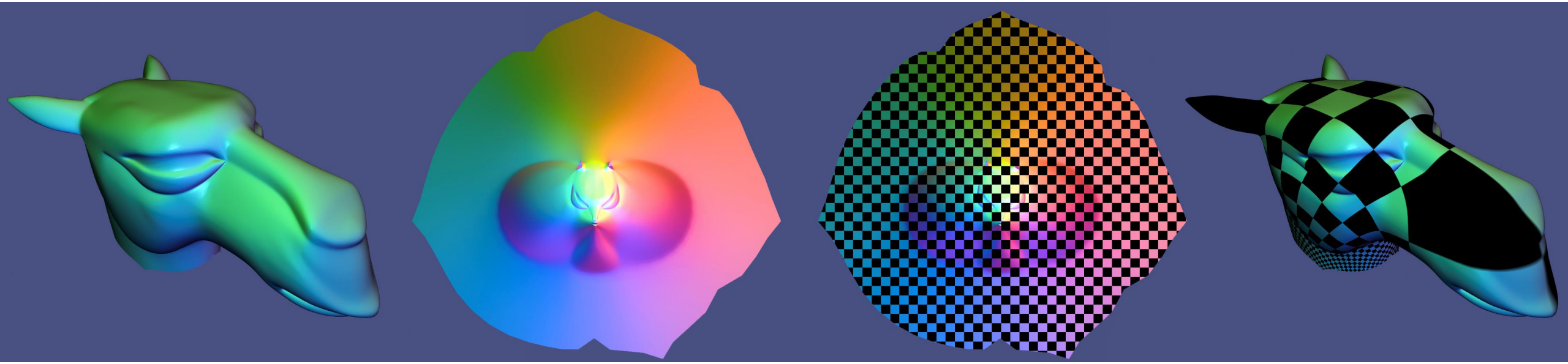
$$\phi : S \to T$$
$$: (x, y, z) \mapsto (u, v)$$



What makes a good function?

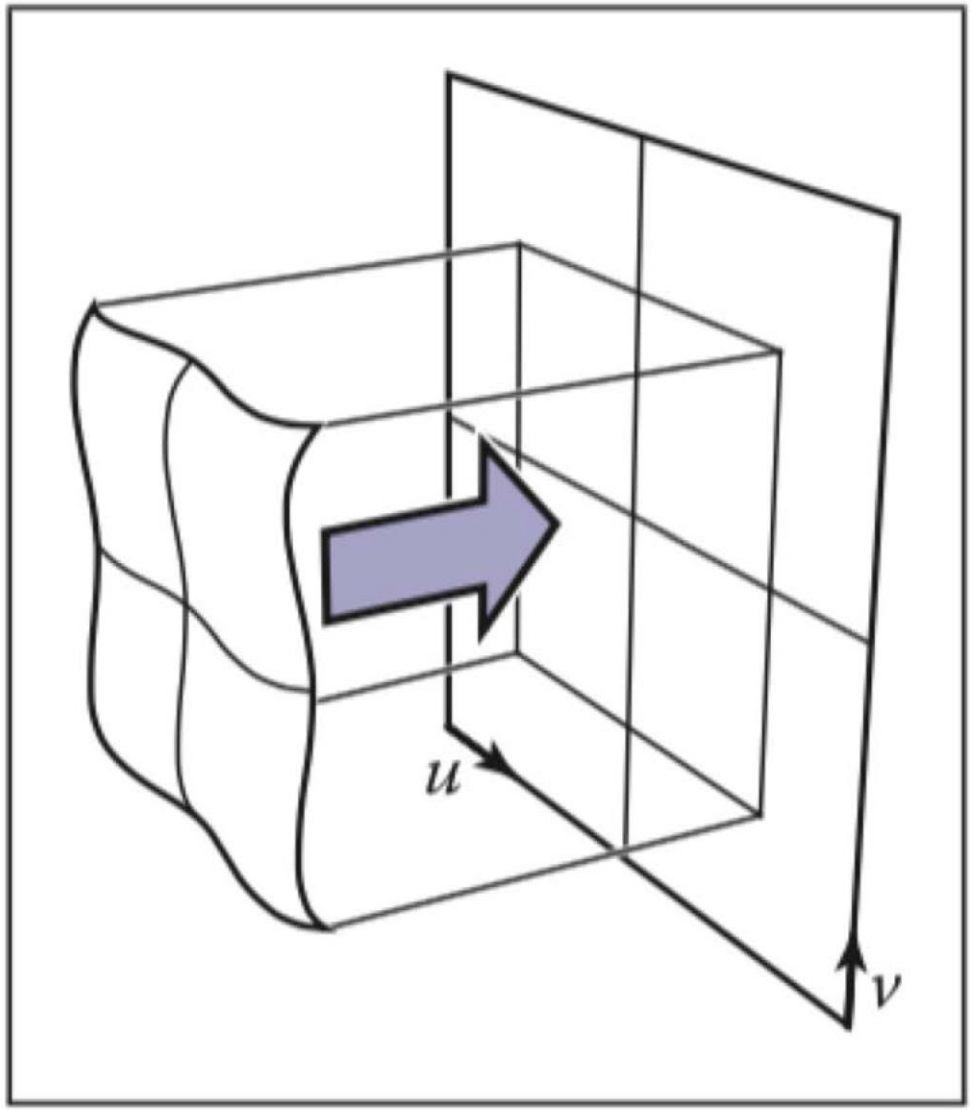
- Bijectivity
- Size distortion
- Shape distortion
- Continuity

$$\phi : S \to T$$
$$: (x, y, z) \mapsto (u, v)$$



https://github.com/alecjacobson/geometry-processing-csc2520/blob/master/lecture-notes/parameterization-ryan-schmidt.pdf
https://members.loria.fr/Bruno.Levy/papers/LSCM_SIGGRAPH_2002.pdf
https://arxiv.org/pdf/1704.06873.pdf

Planar Texture Map

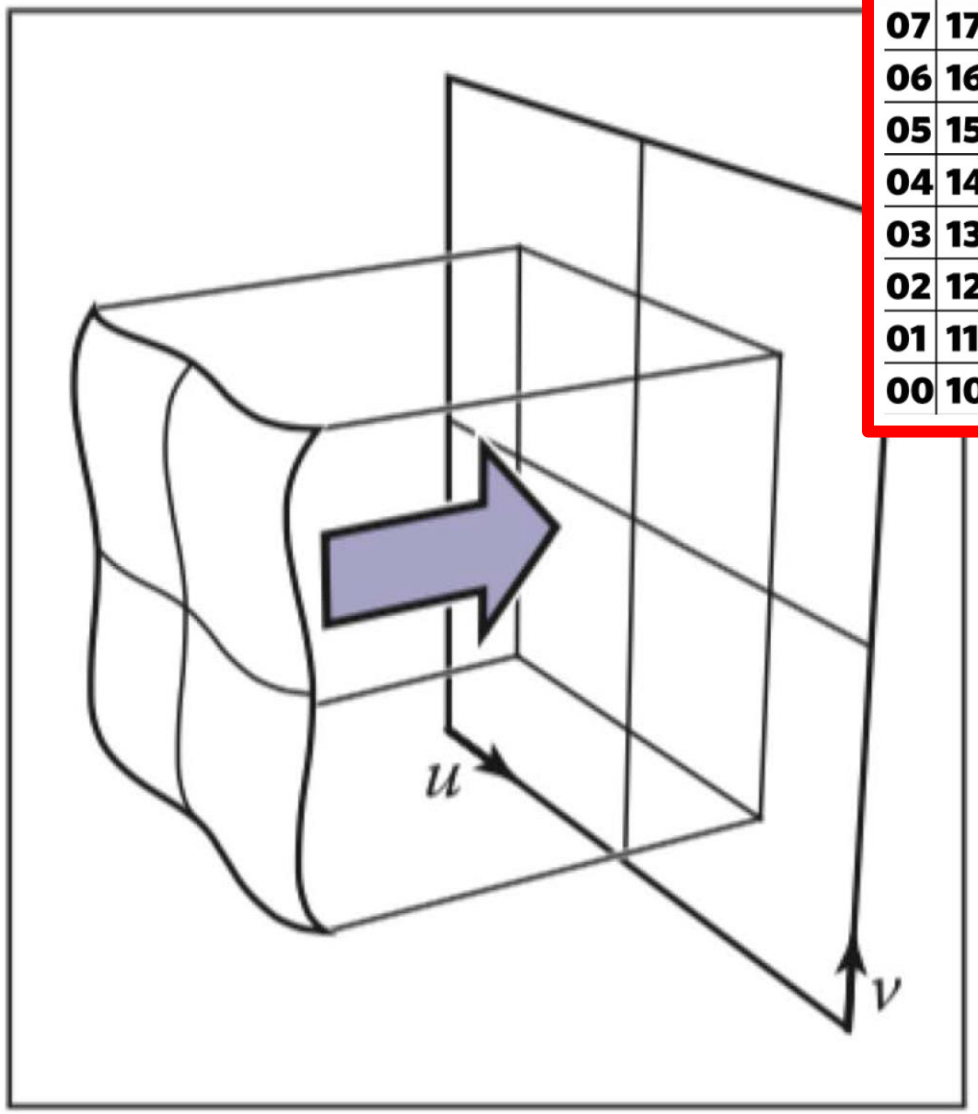


$$\phi(x, y, z) = (u, v)$$

where $$\begin{bmatrix} u \\ v \\ * \\ 1 \end{bmatrix} = M_t \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Planar Texture Map

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90



Spherical Texture Map

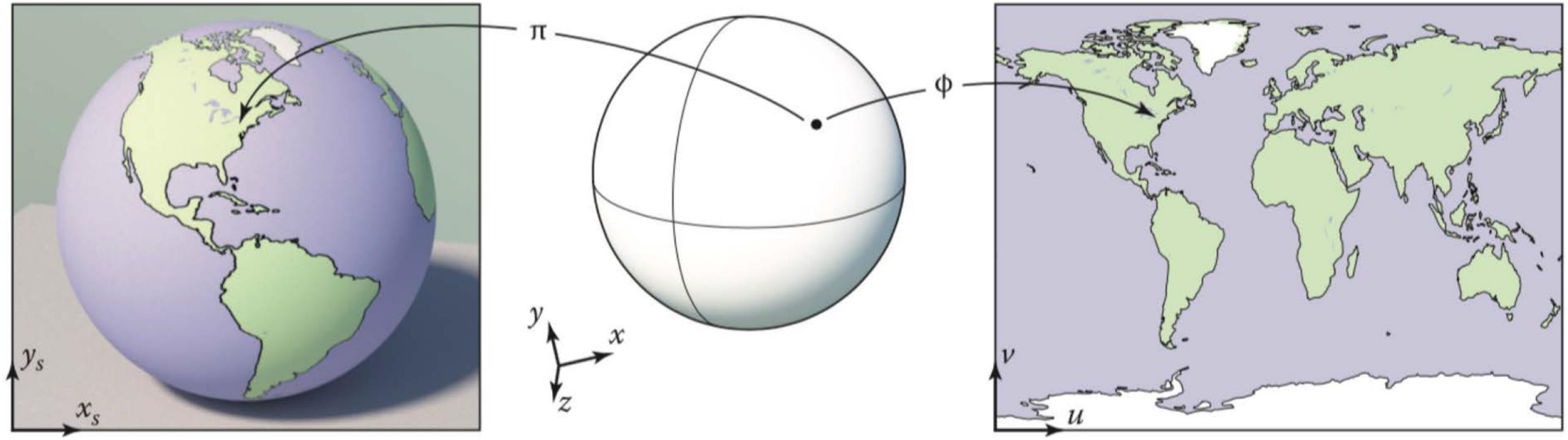
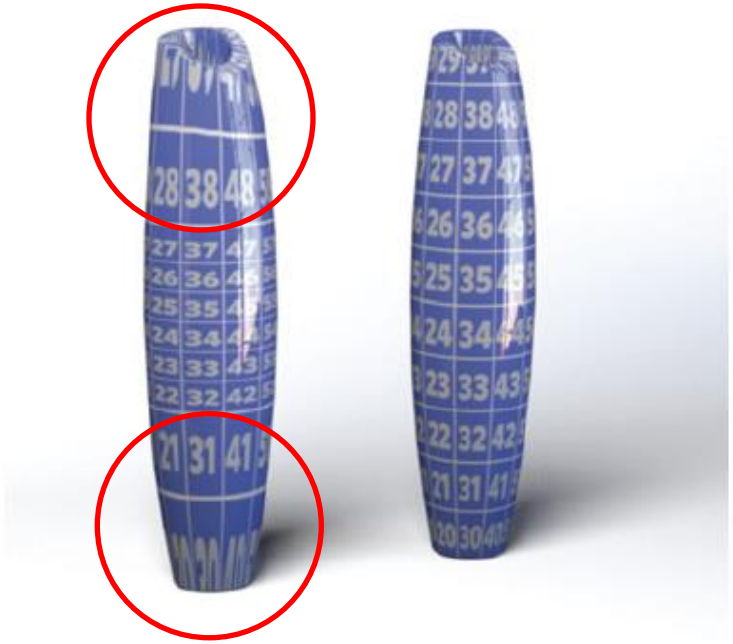


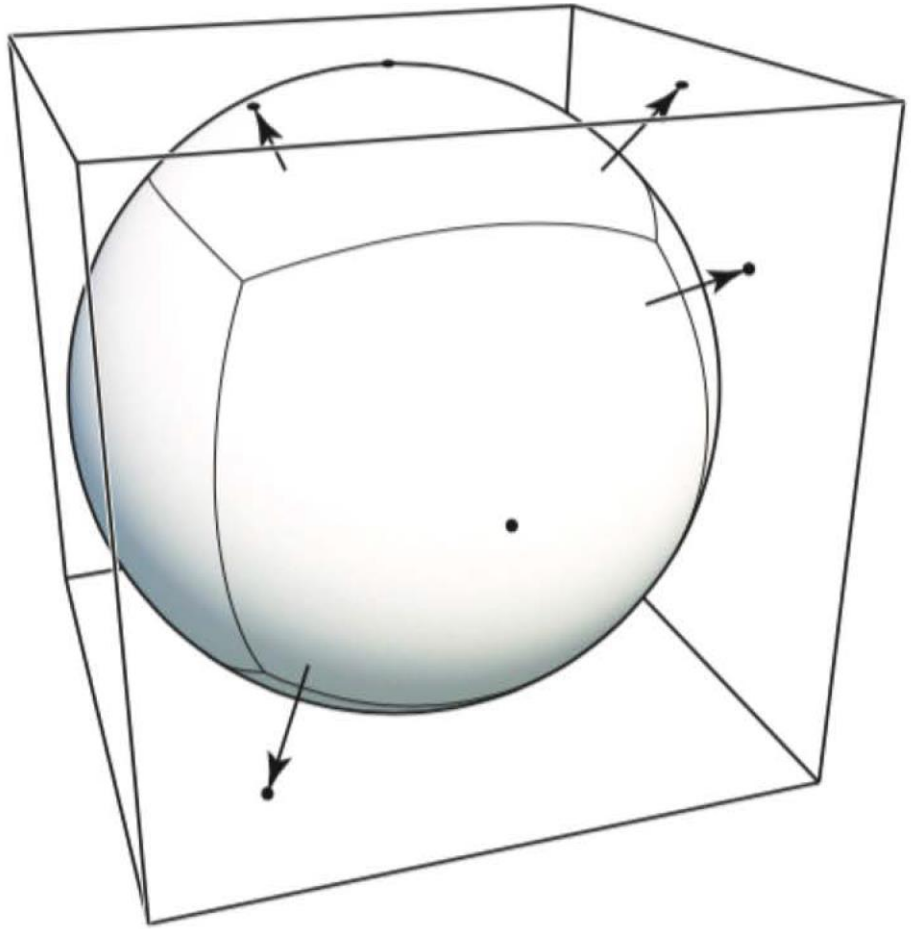
Image space | Surface S in world space | Texture space, T

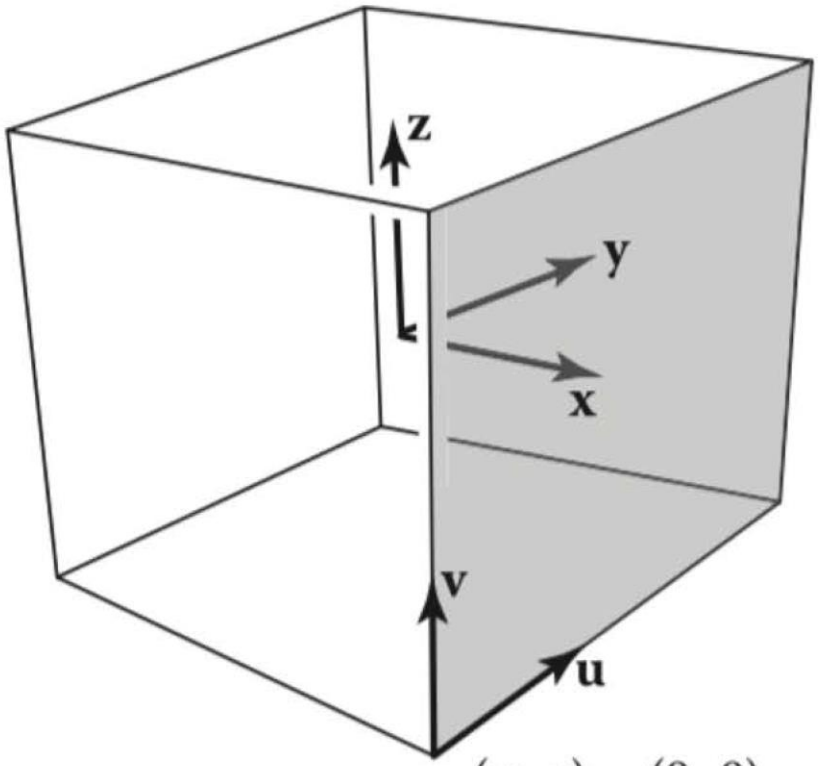
$$\phi(x, y, z) = ([\pi + \text{atan2}(y, x)]/2\pi, [\pi - \text{acos}(z/\|x\|)]/\pi)$$

Spherical Texture Map Distortion



Cube Texture Map





$(u, v) = (1, 1)$
$x = y = z$

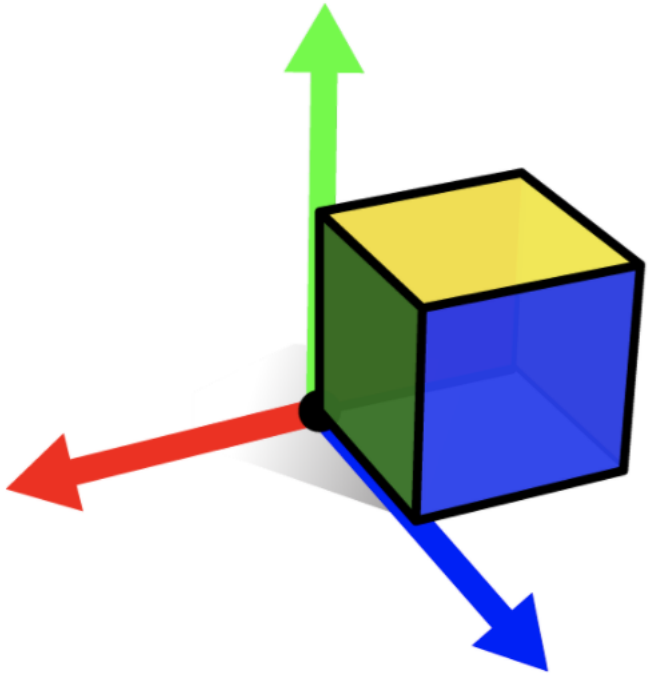
Right face has
$x > |y|$ and
$x > |z|$

$(u, v) = (0, 0)$
$x = -y = -z$

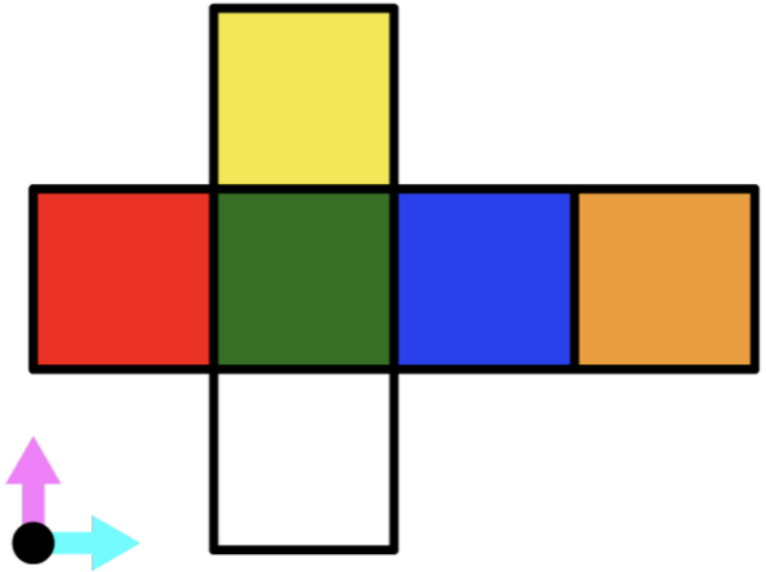
$$(x, y, z) \mapsto \left(\frac{x}{z}, \frac{y}{z}\right).$$

Cube Texture Map

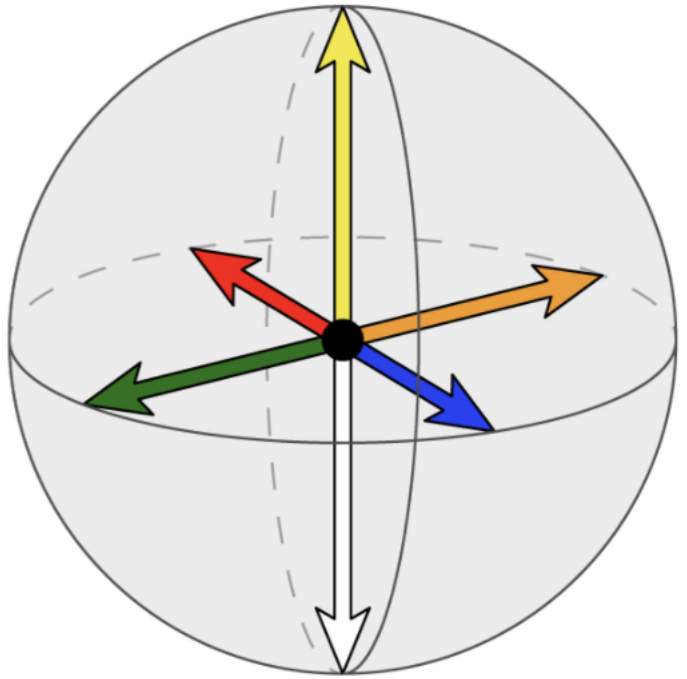
3D Vertex Positions



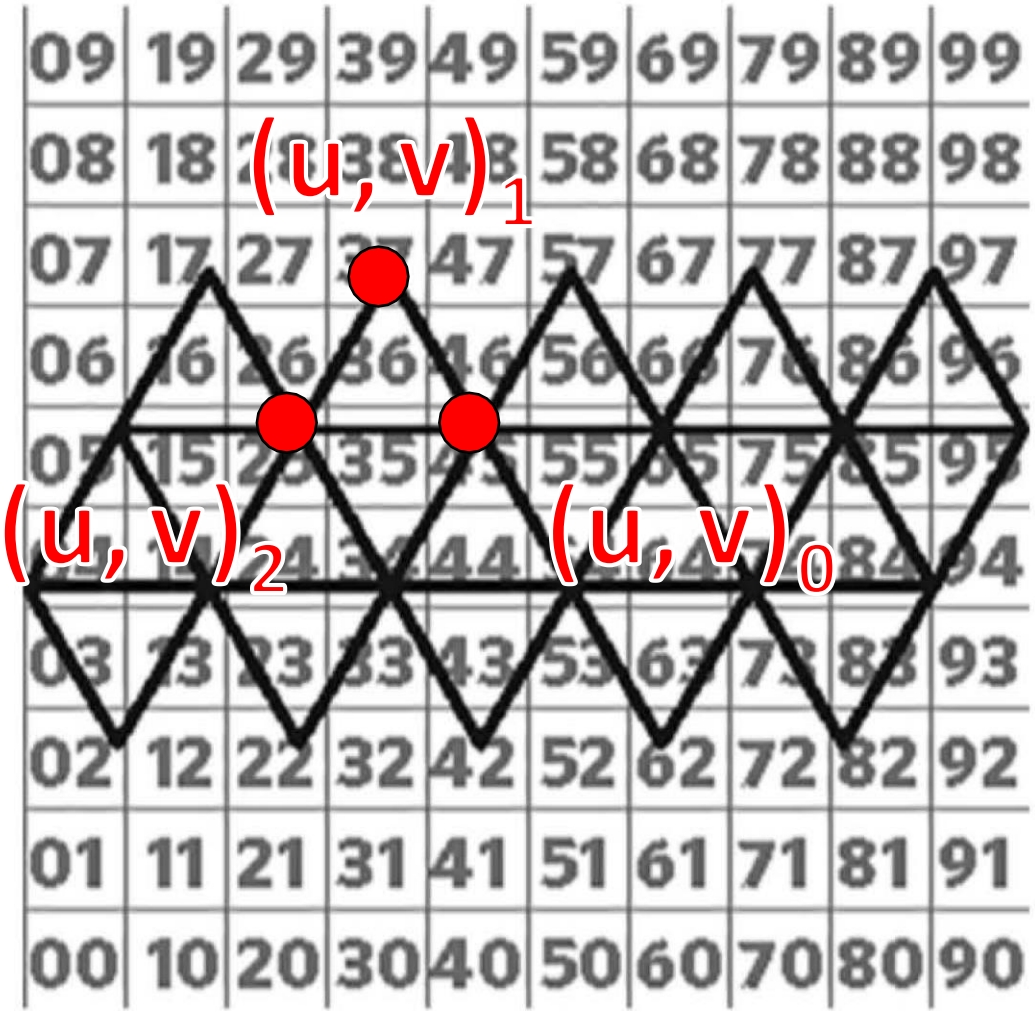
2D Parameterization Positions

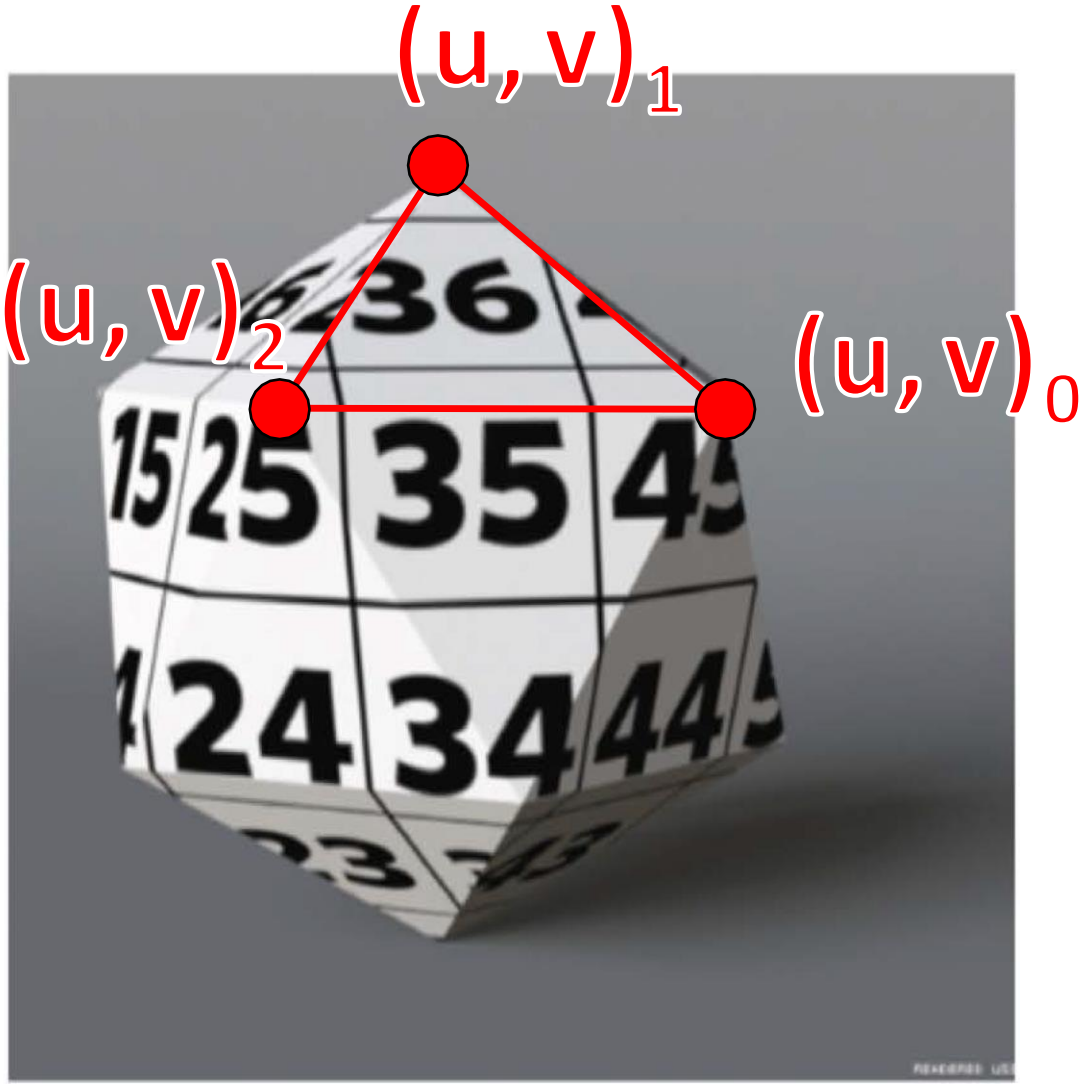


3D Normal Vectors

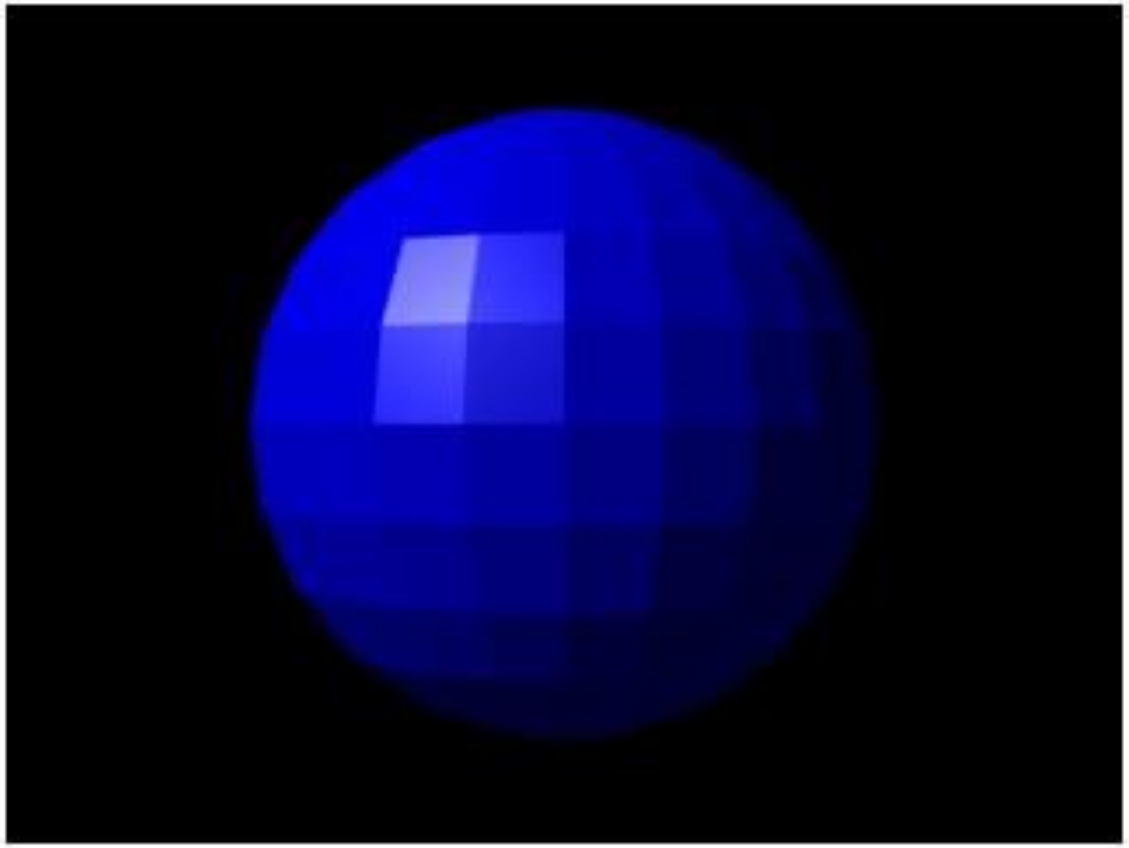


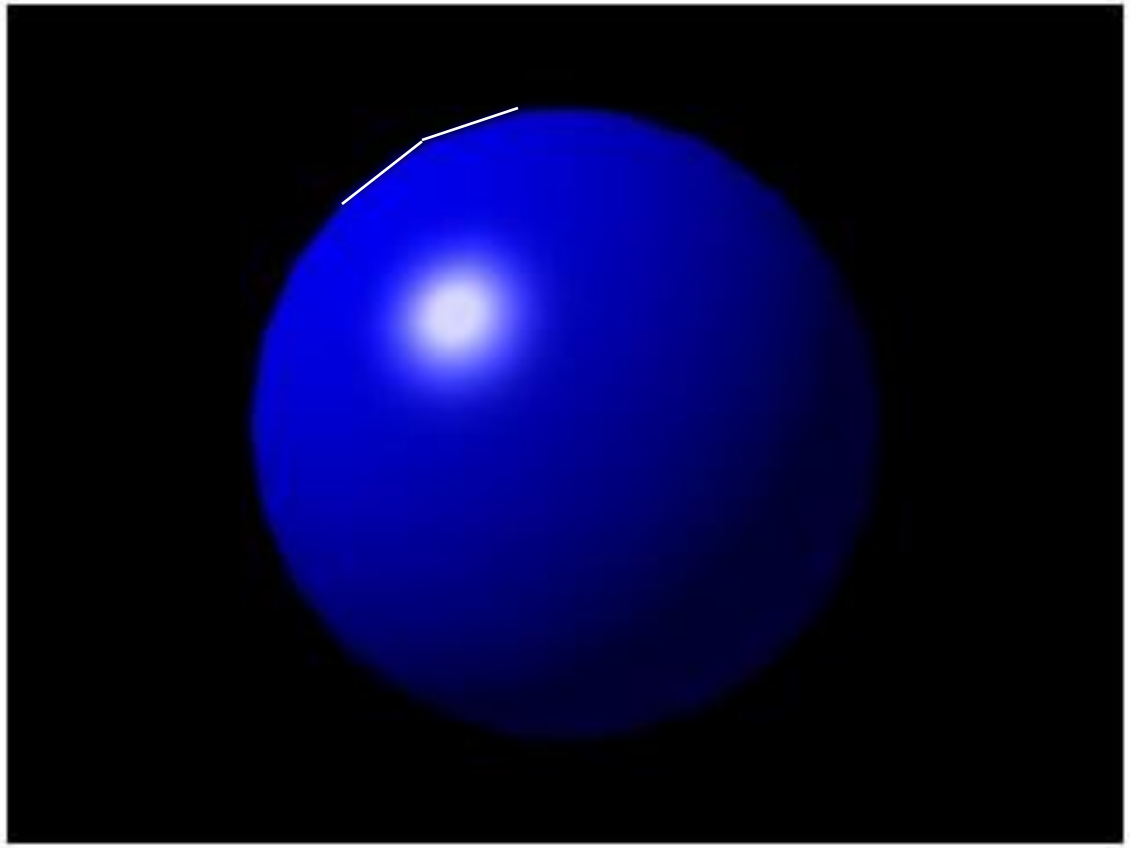
Interpolated Texture Coordinates





Meshes representing smooth surfaces

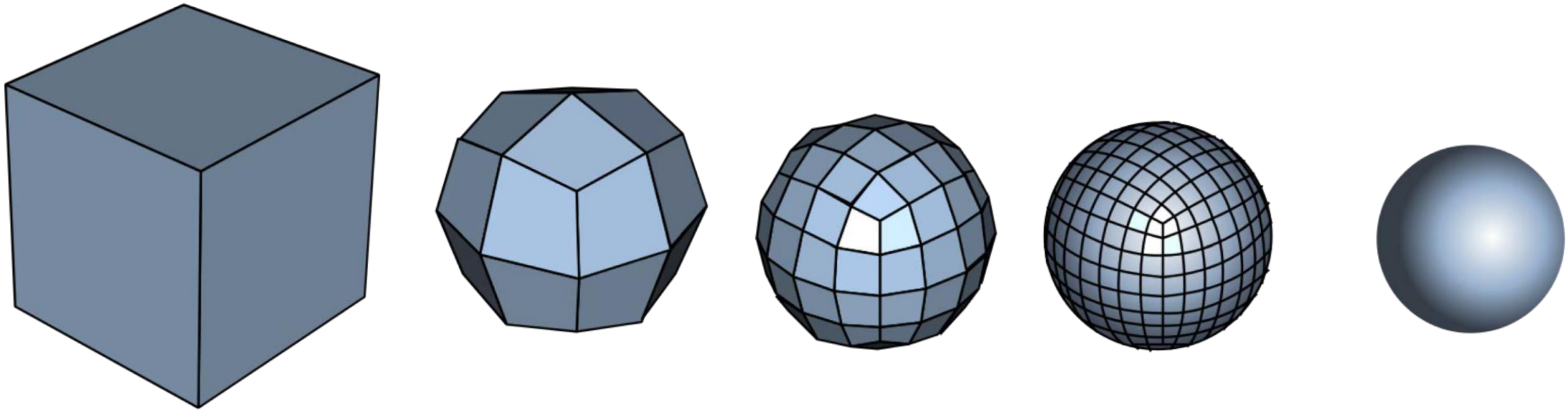




Subdivision Surfaces

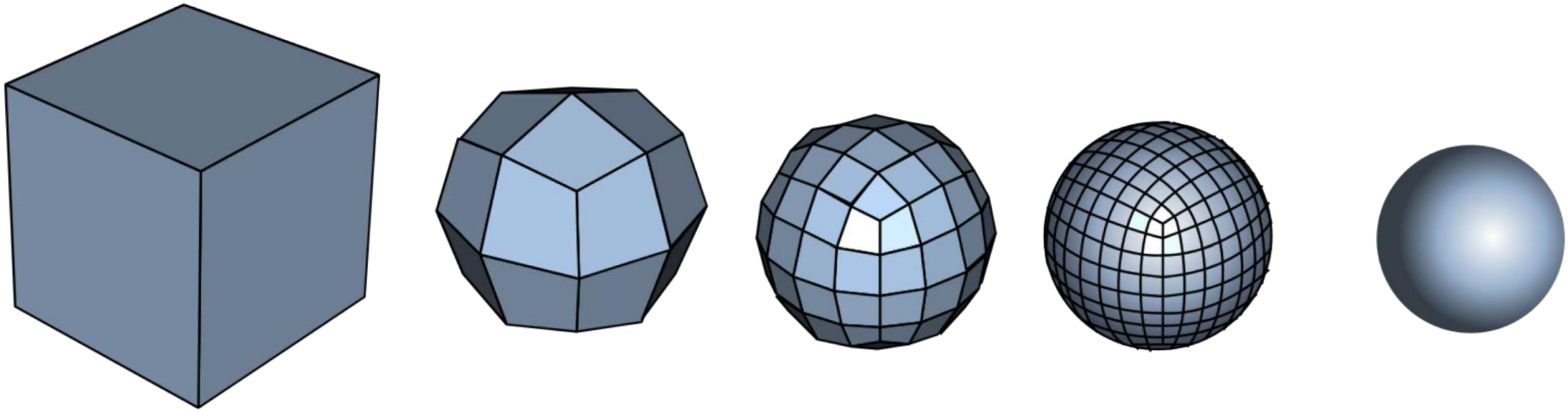
Recursive refinement of polygonal mesh

Results in a smooth “limit surface”



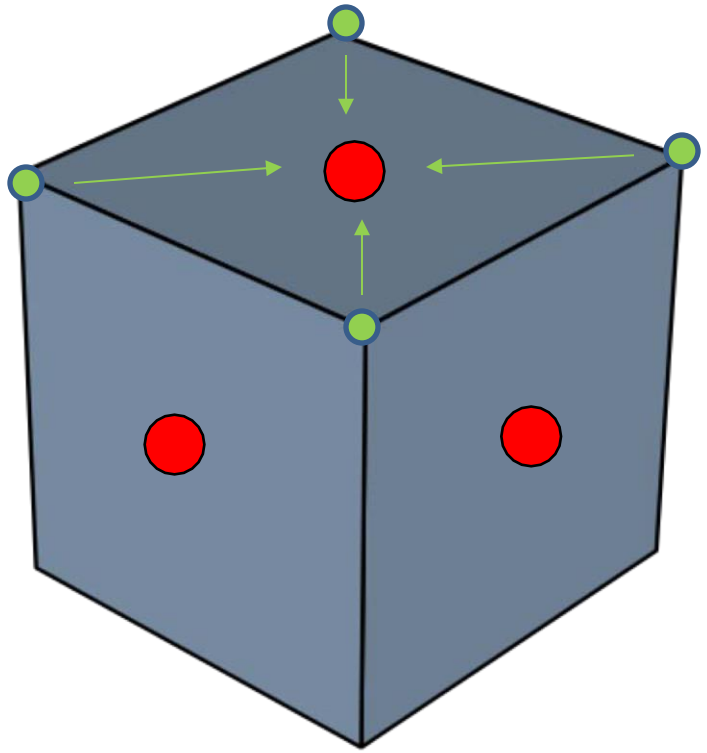
Catmull-Clark Subdivision

Particular type of subdivision scheme.



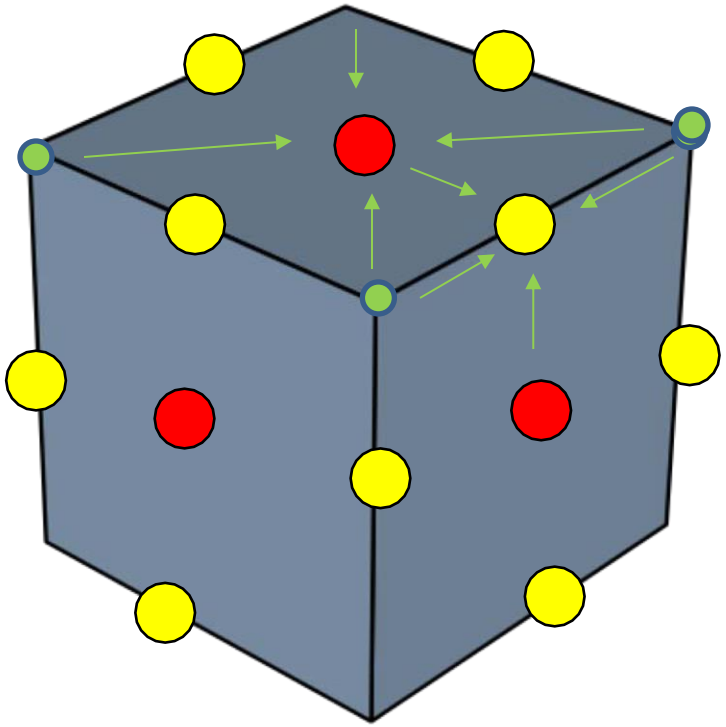
Catmull-Clark Subdivision

Step 1: Set the face point for each facet to be the average of its vertices



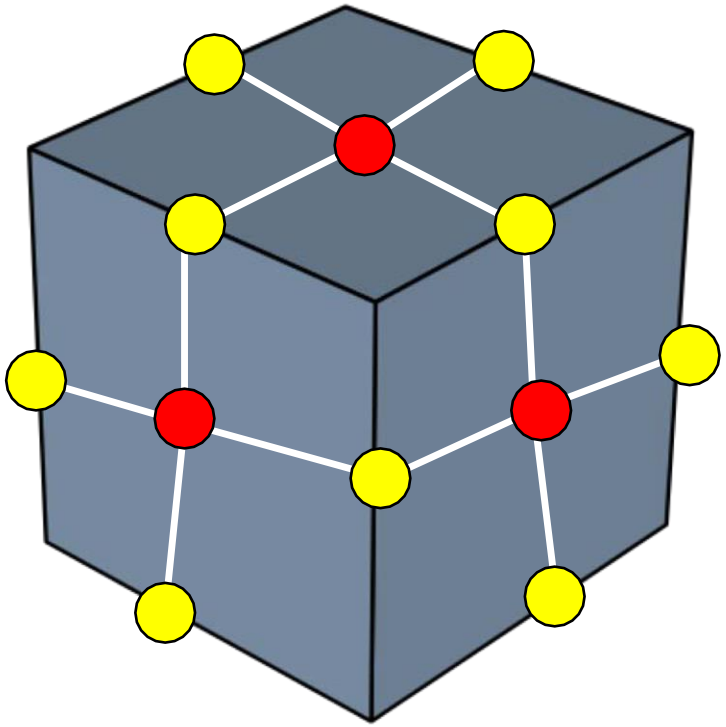
Catmull-Clark Subdivision

Step 2: Add edge points – average of two neighbouring face points and edge end points



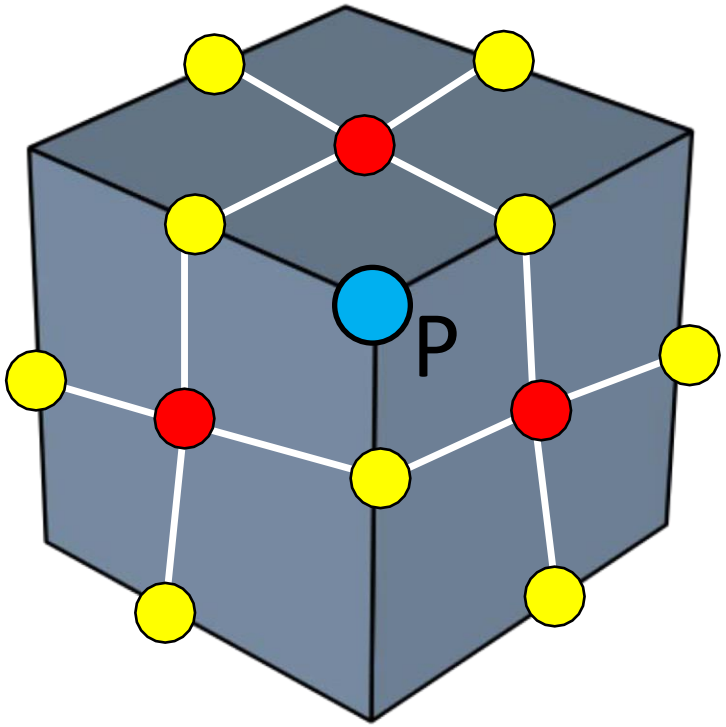
Catmull-Clark Subdivision

Step 3: Add edges between face points and edge points



Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



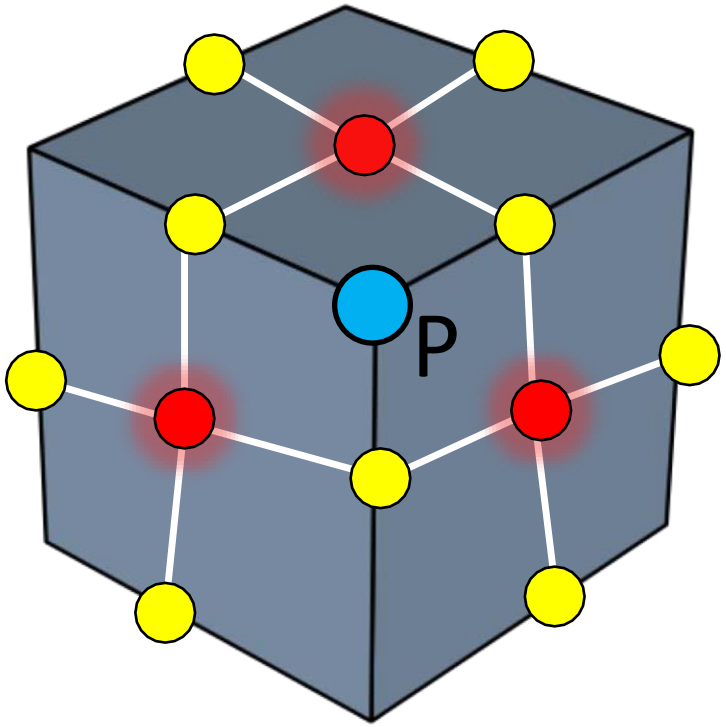
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



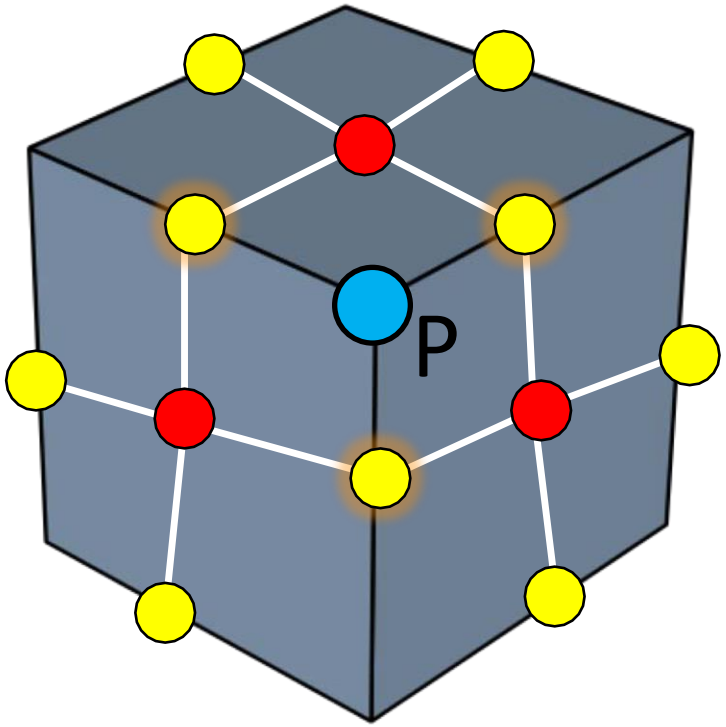
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



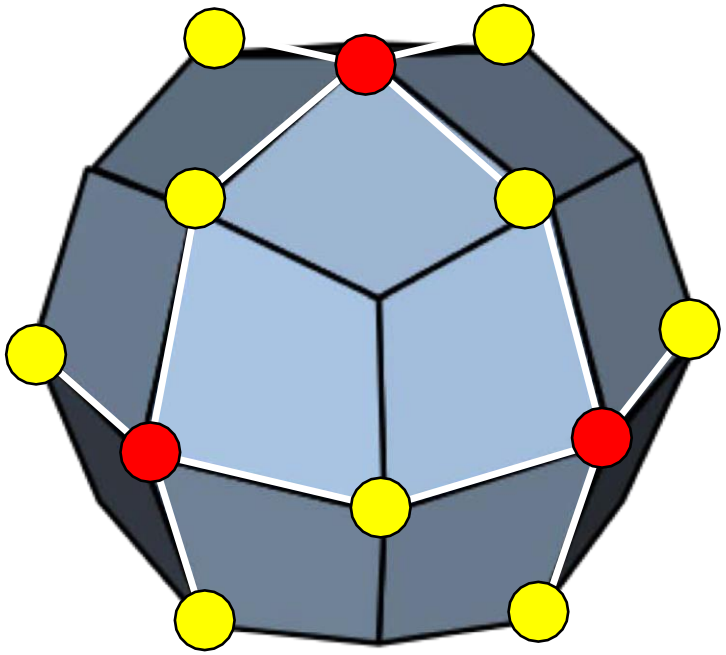
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

Catmull-Clark Subdivision

Step 4: Move each original vertex according to new position given by:



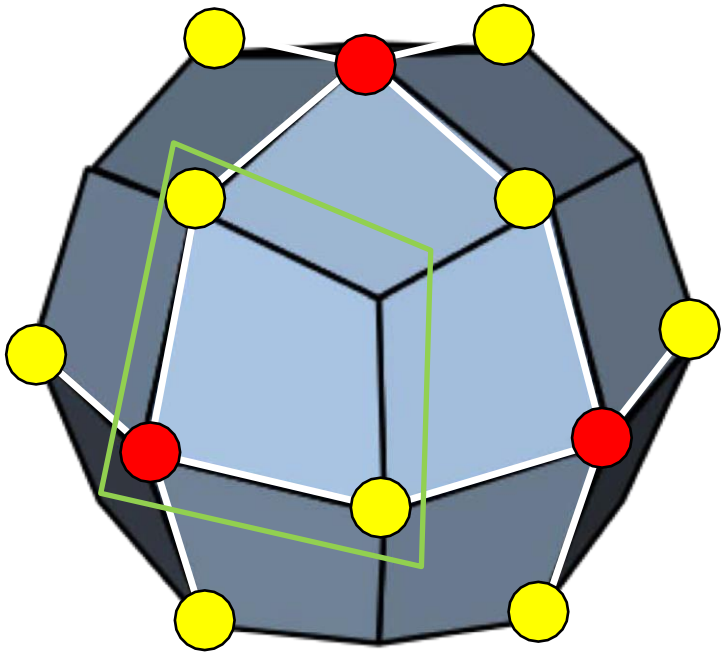
$$\frac{F + 2R + (n-3)P}{n}$$

F : Average of all n created face points adjacent to P

R : Average of all original edge midpoints touching P

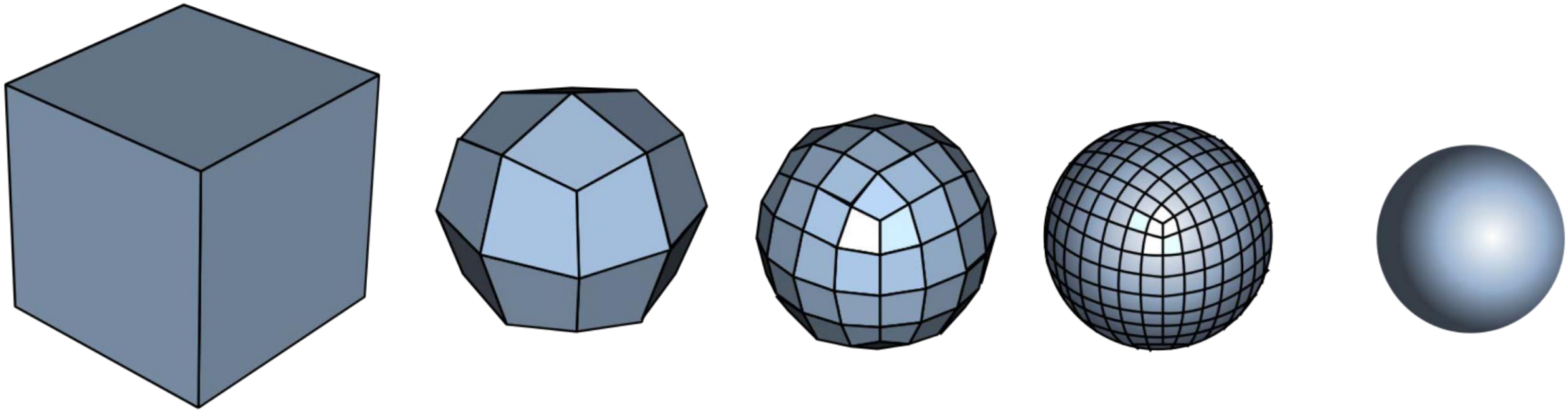
Catmull-Clark Subdivision

Step 5: Connect up original points to make facets



Catmull-Clark Subdivision

Particular type of subdivision scheme.



Subdivision Surfaces in Action

http://graphics.pixar.com/opensubdiv/

Subdivision Surfaces in Action

Figure 5: Geri's hand as a piecewise smooth Catmull-Clark surface. Infinitely sharp creases are used between the skin and the finger nails.

https://www.youtube.com/watch?v=9IYRC7g2ICg
https://graphics.pixar.com/library/Geri/paper.pdf

Where to find meshes?

Make your own!

MAYA

https://www.blender.org/

https://www.autodesk.ca/en/products/maya/overview

Download them!

https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/

https://www.turbosquid.com/

https://poly.google.com/

https://www.thingiverse.com/

https://ten-thousand-models.appspot.com/

Assignment #5: Tasks

- src/write_obj.cpp Write a pure-triangle or pure-quad mesh with 3D vertex positions V and faces F, 2D parametrization positions UV and faces UF, 3D normal vectors NV and faces NF to a .obj file. Note: These *two* function overloads represent only a small subset of meshes and mesh-data that can be written to a .obj file.
- src/cube.cpp Construct the quad mesh of a cube including parameterization and per-face normals.
  Hint: Draw out on paper and *label* with indices the 3D cube, the 2D parameterized cube, and the normals.
- src/sphere.cpp Construct a quad mesh of a sphere with num_faces_u × num_faces_v faces.
- src/triangle_area_normal.cpp Compute the normal vector of a 3D triangle given its corner locations. The output vector should have length equal to the area of the triangle.
- src/per_face_normals.cpp Compute per-face normals for a triangle mesh.
- src/per_vertex_normals.cpp Compute per-vertex normals for a triangle mesh.
- src/vertex_triangle_adjacency.cpp Compute a vertex-triangle adjacency list. For each vertex store a list of all incident faces.
- src/per_corner_normals.cpp Compute per corner normals for a triangle mesh by computing the area-weighted average of normals at incident faces whose normals deviate less than the provided threshold.
- src/catmull_clark.cpp Conduct num_iters iterations of Catmull-Clark subdivision on a **pure quad** mesh (V,F).

Next: Transformations and Shaders