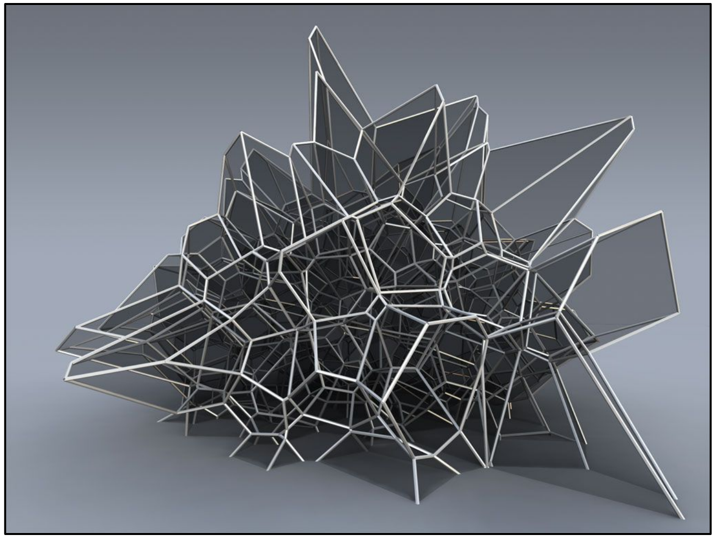
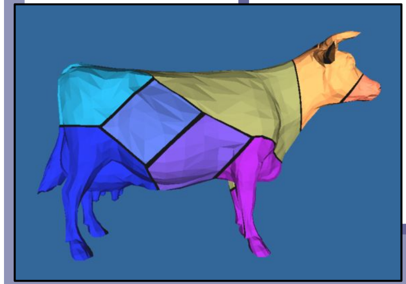
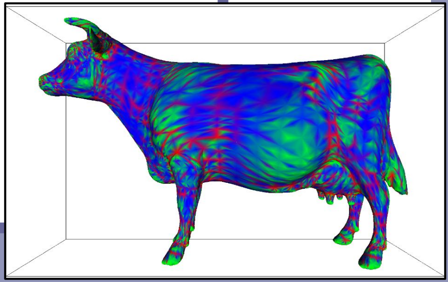
Further Graphics



A Brief Introduction to Computational Geometry



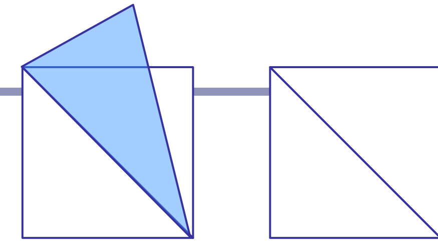


Alex Benton, University of Cambridge – alex@bentonian.com

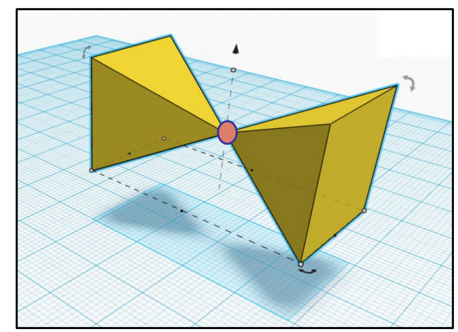
Supported in part by Google UK, Ltd

Computational Geometry

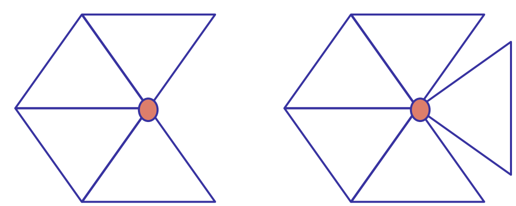
- Polygons meshes are examples of *discrete* (as opposed to continuous) representation of geometry
  - Many rendering systems limit themselves to triangle meshes
  - Many require that the mesh be *manifold*
- In a *closed manifold* polygon mesh:
  - Exactly two triangles meet at each edge
  - The faces meeting at each vertex belong to a single, connected loop of faces
- In a *manifold with boundary*:
  - At most two triangles meet at each edge
  - The faces meeting at each vertex belong to a single, connected strip of faces



Edge: Non-manifold vs manifold



Non-manifold vertex

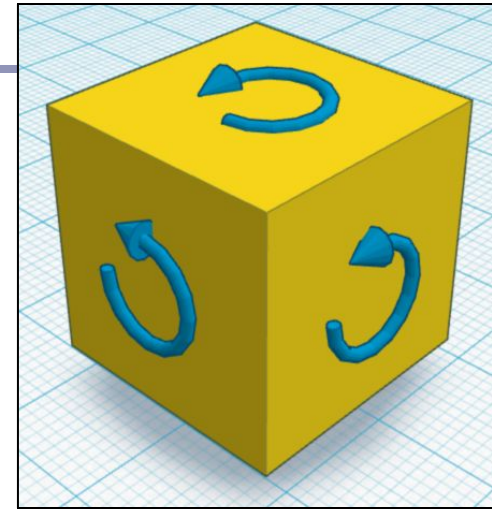


Vertex: Good boundary vs bad

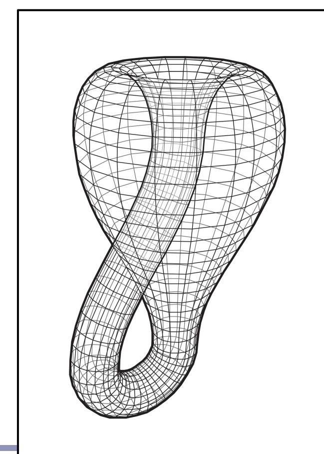
This slide draws much inspiration from Shirley and Marschner's *Fundamentals of Computer Graphics*, pp. 262-263

Terminology

- We say that a surface is *oriented* if:
    a. the vertices of every face are stored in a fixed order
    b. if vertices *i*, *j* appear in both faces *f1* and *f2*, then the vertices appear in order *i*, *j* in one and *j*, *i* in the other
- We say that a surface is *embedded* if, informally, "nothing pokes through":
    a. No vertex, edge or face shares any point in space with any other vertex, edge or face except where dictated by the data structure of the polygon mesh
- A closed, embedded surface must separate 3-space into two parts: a bounded *interior* and an unbounded *exterior*.



A cube with "anti-clockwise" oriented faces



Klein bottle: not an embedded surface.

Also, terrible for holding drinks.

Gaussian curvature on smooth surfaces

Informally speaking, the *curvature* of a surface expresses "how flat the surface isn't".

- One can measure the directions in which the surface is curving *most*; these are the directions of *principal curvature*, k_1 and k_2.
- The product of k_1 and k_2 is the scalar *Gaussian curvature*.

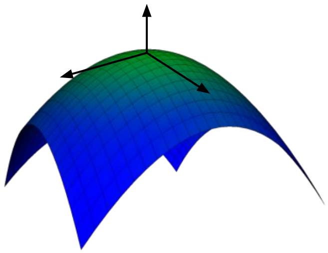
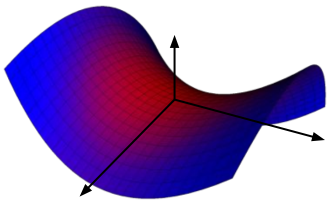
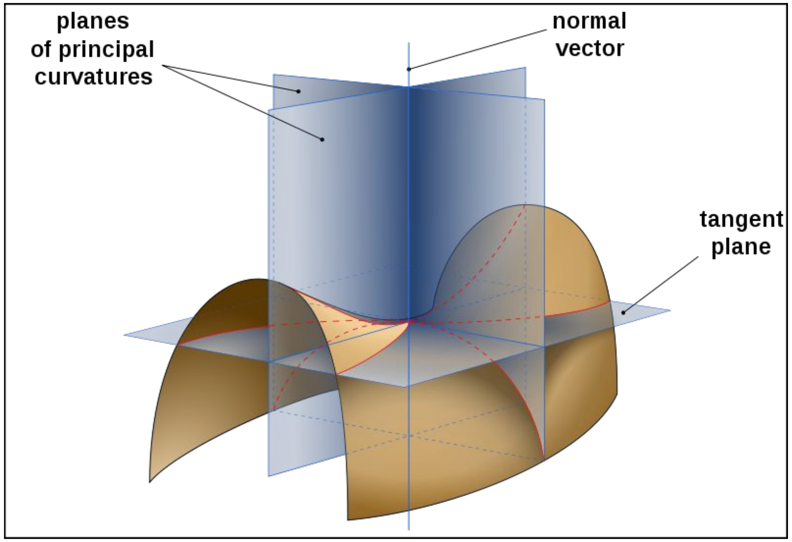
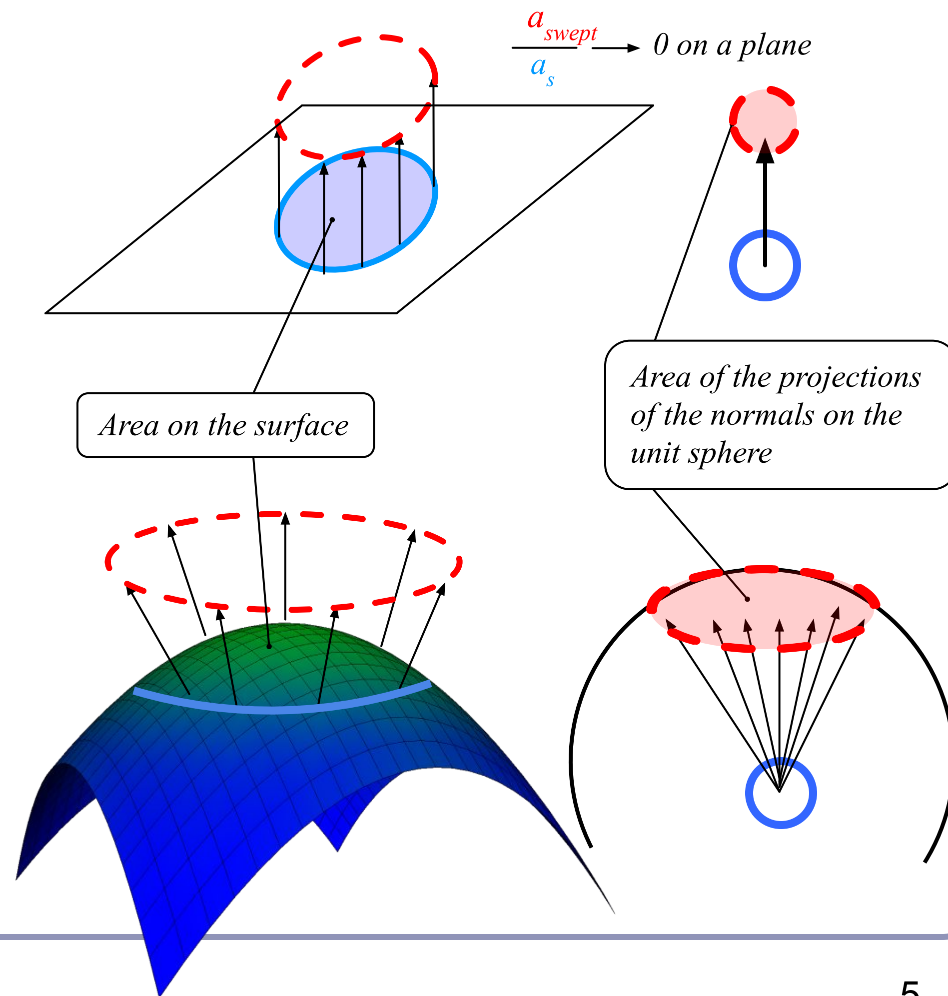




Image by Eric Gaba, from Wikipedia

Gaussian curvature on smooth surfaces

Formally, the *Gaussian curvature of a region* on a surface is the ratio between the area of the surface of the unit sphere swept out by the normals of that region and the area of the region itself.

The Gaussian curvature of a point is the limit of this ratio as the region tends to zero area.



$\frac{a_{swept}}{a_s} \rightarrow r^{-2}$ *on a sphere of radius r*
(please pretend that this is a sphere)

Gaussian curvature on discrete surfaces

On a discrete surface, normals do not vary smoothly: the normal to a face is constant on the face, and at edges and vertices the normal is—strictly speaking—undefined.

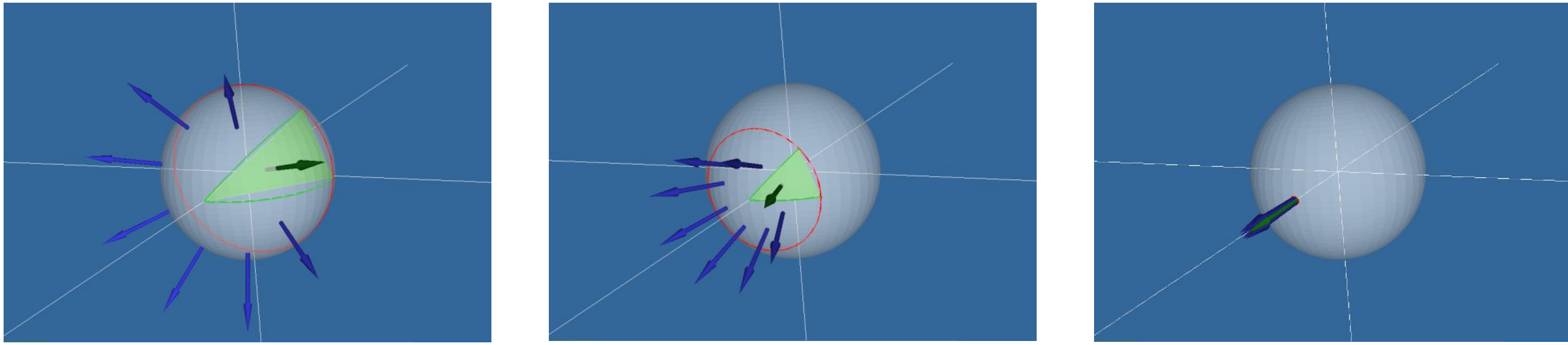
- Normals change instantaneously (as one's point of view travels across an edge from one face to another) or not at all (as one's point of view travels within a face.)

The Gaussian curvature of the surface of any polyhedral mesh is **zero** everywhere except at the vertices, where it is **infinite**.

Normal on a surface

Expressed as a limit,

The *normal of surface S at point P* is the limit of the cross-product between two (non-collinear) vectors from *P* to the set of points in *S* at a distance *r* from *P* as *r* goes to zero. [Excluding orientation.]



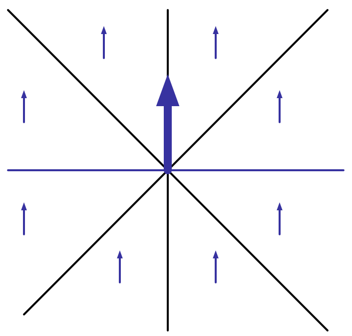
Normal at a vertex

Using the limit definition, is the 'normal' to a discrete surface necessarily a vector?

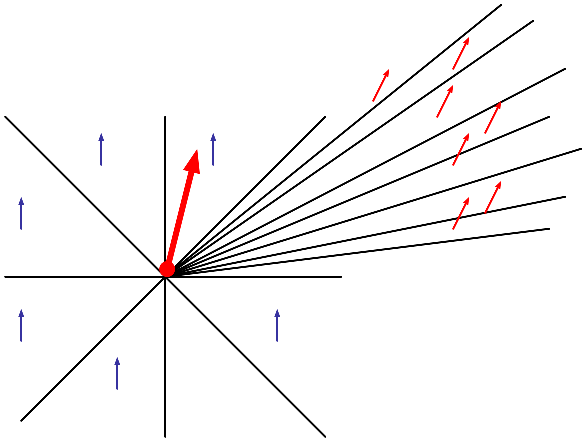
- The normal to the surface at any point on a face is a constant vector.
- The 'normal' to the surface at any edge is an arc swept out on a unit sphere between the two normals of the two faces.
- The 'normal' to the surface at a vertex is a space swept out on the unit sphere between the normals of all of the adjacent faces.

Finding the normal at a vertex

Method 1: Take the average of the normals of surrounding polygons



Problem: splitting one adjacent face into 10,000 shards would skew the average



Finding the normal at a vertex

Method 2: Take the weighted average of the normals of surrounding polygons, weighted by the area of each face

- 2a: Weight each face normal by the area of the face divided by the total number of vertices in the face

Problem: Introducing new edges into a neighboring face (and thereby reducing its area) should not change the normal.

Should making a face larger affect the normal to the surface near its corners?

- Argument for yes: If the vertices interpolate the 'true' surface, then stretching the surface at a distance could still change the local normals.

Finding the normal at a vertex

Method 3: Take the weighted average of the normals of surrounding polygons, weighted by each polygon's *face angle* at the vertex

Face angle: the angle α formed at the vertex *v* by the vectors to the next and previous vertices in the face *F*

$$\alpha(F, v_i) = cos^{-1}\left(\frac{v_{i+1} - v_i}{|v_{i+1} - v_i|} \bullet \frac{v_{i-1} - v_i}{|v_{i-1} - v_i|}\right)$$

$$N(v) = \frac{\sum_F \alpha(F, v)\, N_F}{|\sum_F \alpha(F, v)|}$$

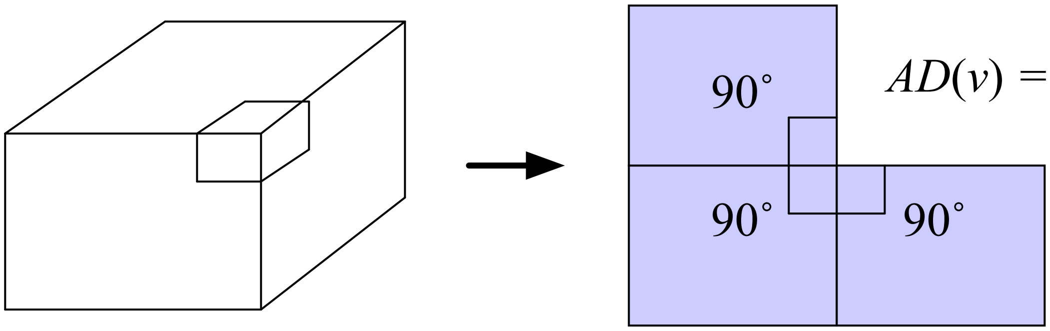
Note: In this equation, *arccos* implies a convex polygon. Why?

Angle deficit – a better solution for measuring discrete curvature

The *angle deficit AD(v)* of a vertex *v* is defined to be two π minus the sum of the *face angles α(F)* of the adjacent faces

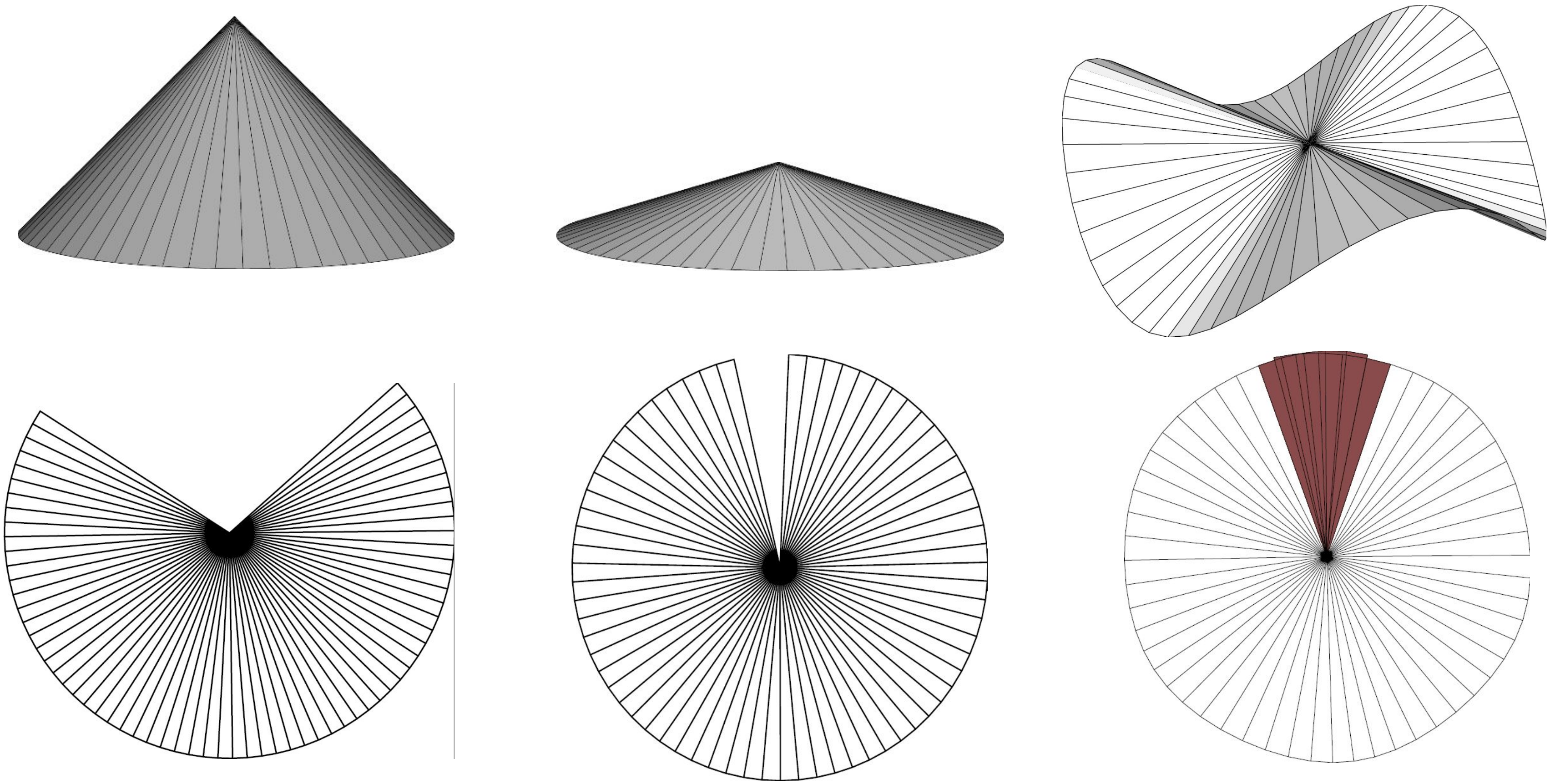
$$\alpha(F, v_i) = cos^{-1}\left(\frac{v_{i+1} - v_i}{|v_{i+1} - v_i|} \bullet \frac{v_{i-1} - v_i}{|v_{i-1} - v_i|}\right)$$

$$AD(v) = 2\pi - \sum_F \alpha(F, v)$$



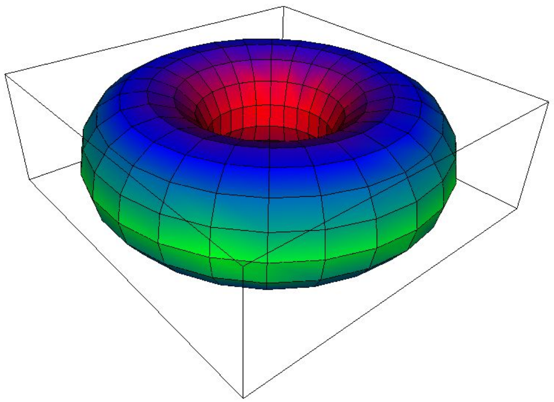
$AD(v) = 360° – 270° = 90°$

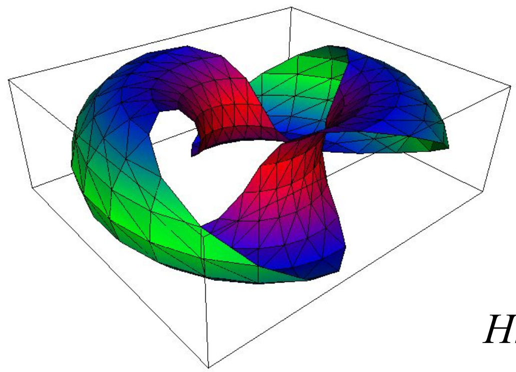
Angle deficit

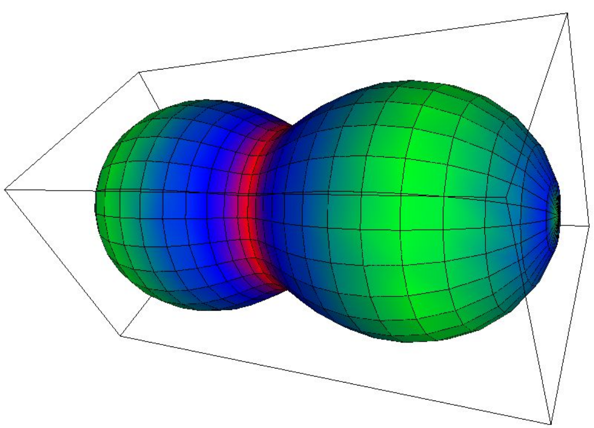


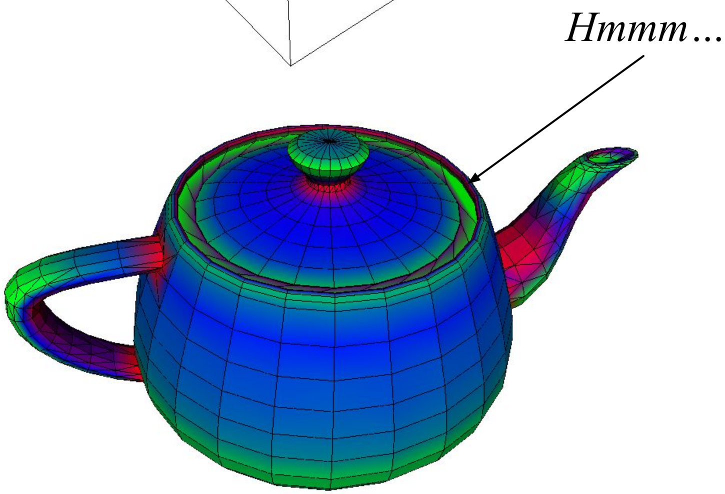
High angle deficit Low angle deficit Negative angle deficit

Angle deficit









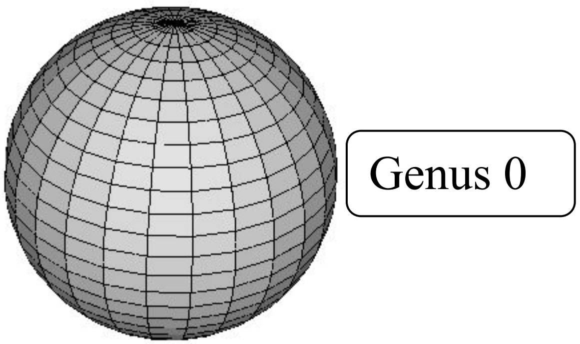
Genus, Poincaré and the Euler Characteristic

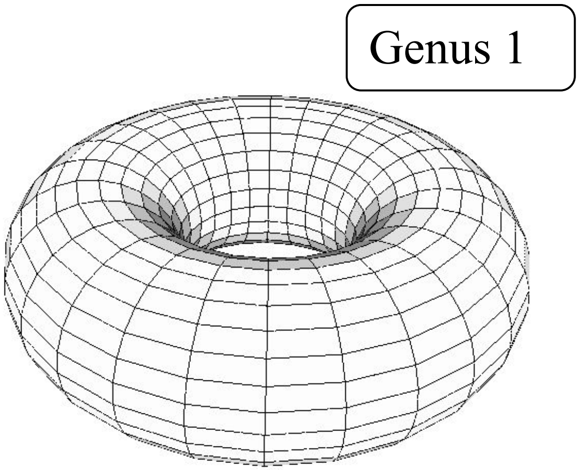
- Formally, the *genus g* of a closed surface is

  ..."a topologically invariant property of a surface defined as the largest number of nonintersecting simple closed curves that can be drawn on the surface without separating it."

  --mathworld.com

- Informally, it's the number of coffee cup handles in the surface.





Genus, Poincaré and the Euler Characteristic

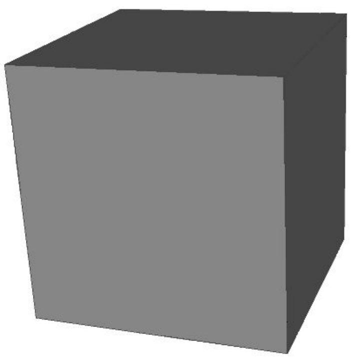
Given a polyhedral surface *S* without border where:

- *V* = the number of vertices of *S*,
- *E* = the number of edges between those vertices,
- *F* = the number of faces between those edges,
- χ is the *Euler Characteristic* of the surface,

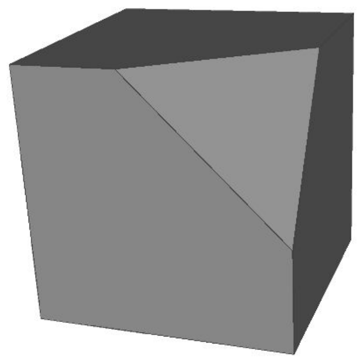
the Poincaré Formula states that:

$$V - E + F = 2 - 2g = \chi$$

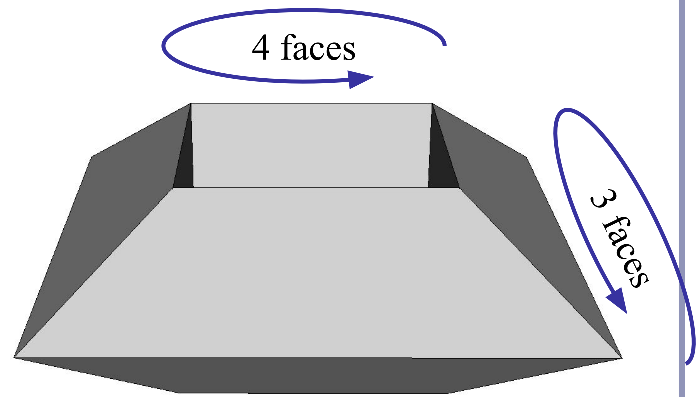
Genus, Poincaré and the Euler Characteristic



$g = 0$
$E = 12$
$F = 6$
$V = 8$
$\underline{V\text{-}E+F = 2\text{-}2g = 2}$



$g = 0$
$E = 15$
$F = 7$
$V = 10$
$\underline{V\text{-}E+F = 2\text{-}2g = 2}$



$g = 1$
$E = 24$
$F = 12$
$V = 12$
$\underline{V\text{-}E+F = 2\text{-}2g = 0}$

The Euler Characteristic and angle deficit

Descartes' *Theorem of Total Angle Deficit* states that on a surface S with Euler characteristic χ, the sum of the angle deficits of the vertices is $2\pi\chi$:

$$\sum_S AD(v) = 2\pi\chi$$

Cube:

- $\chi = 2\text{-}2g = 2$
- $AD(v) = \pi/2$
- $8(\pi/2) = 4\pi = 2\pi\chi$

Tetrahedron:

- $\chi = 2\text{-}2g = 2$
- $AD(v) = \pi$
- $4(\pi) = 4\pi = 2\pi\chi$

Speed things up!
Bounding volumes

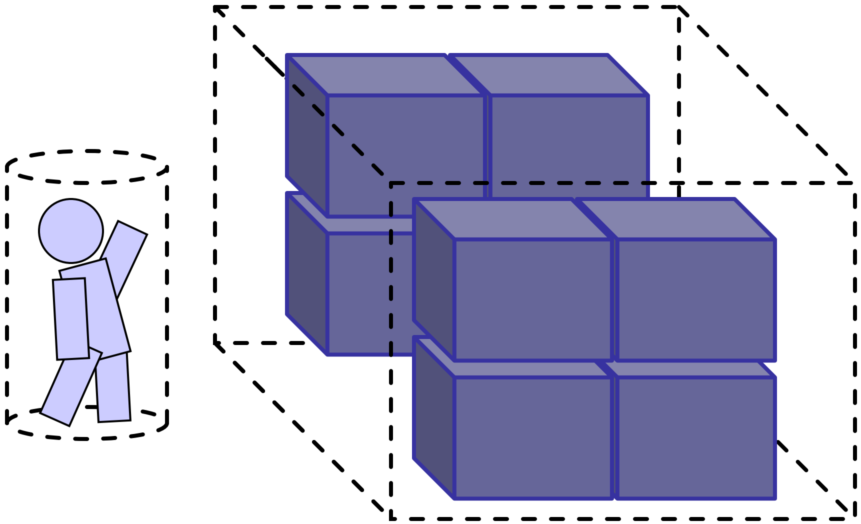
A common optimization method for ray-based rendering is the use of *bounding volumes*.

Nested bounding volumes allow the rapid culling of large portions of geometry

- Test against the bounding volume of the top of the scene graph and then work down.

Great for…

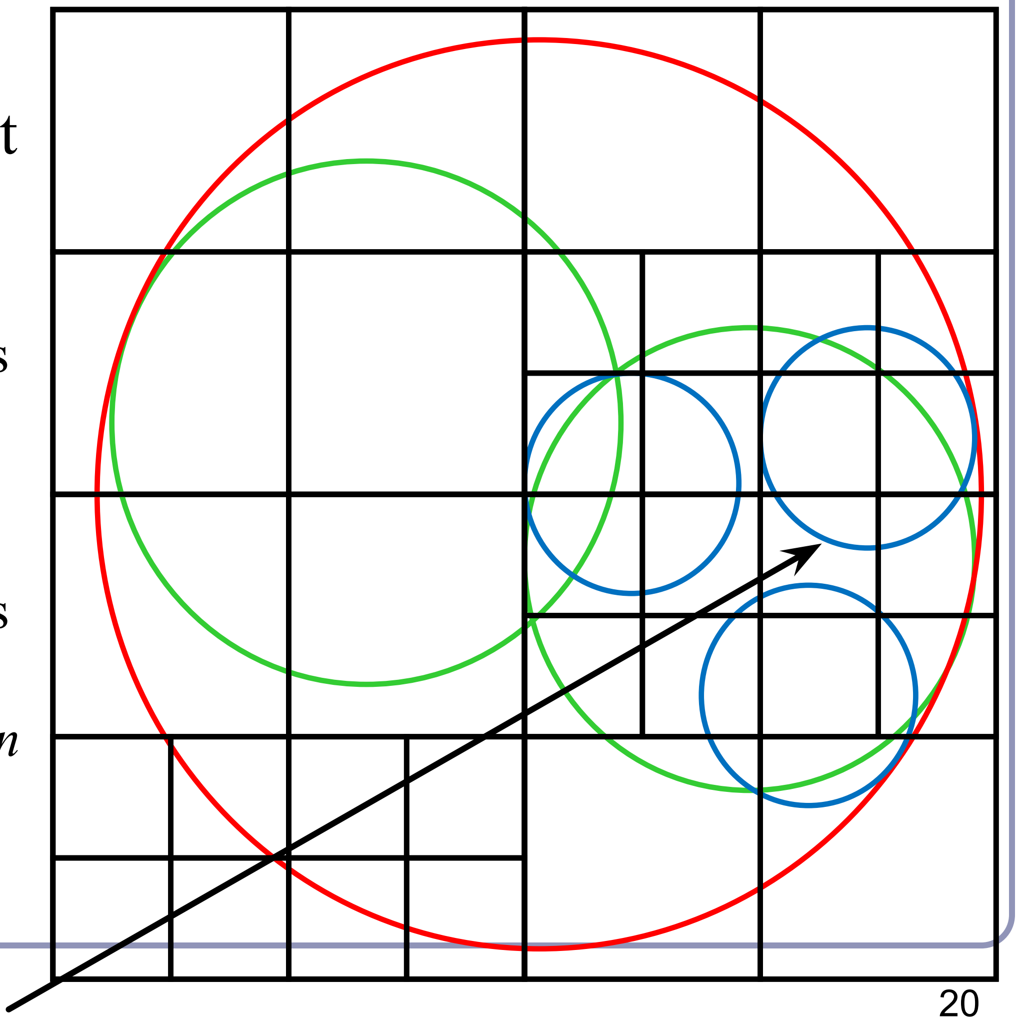
- Collision detection between scene elements
- Culling before rendering
- Accelerating ray-tracing, -marching



Popular acceleration structures: Octrees

Split space into cells and list in each cell every object in the scene that overlaps that cell.

- The ray can skip empty cells
- Requires preprocessing stage, but can be partially updated for moving scenes
- Popular for voxelized games
- The Octree data structure generalizes to arbitrary *n*x*n*x*n* rectangular volume subdivision



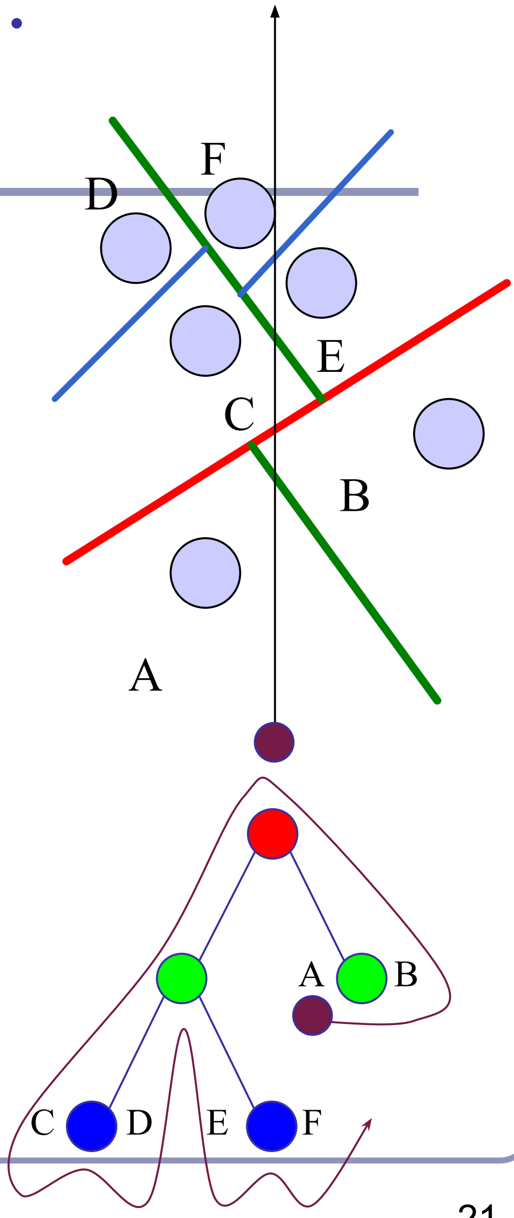
Popular acceleration structures: BSP Trees

The *BSP tree* pre-partitions the scene into objects in front of, on, and behind a tree of planes.

- This gives an ordering in which to test scene objects against your ray
- When you fire a ray into the scene, you test all near-side objects before testing far-side objects.

Challenges:

- requires slow pre-processing step
- strongly favors static scenes
- choice of planes is hard to optimize



Popular acceleration structures: *kd-trees*

The *kd-tree* is a simplification of the BSP Tree data structure

- Space is recursively subdivided by axis-aligned planes and points on either side of each plane are separated in the tree.
- The *k*d-tree has O(*n* log *n*) insertion time (but this is very optimizable by domain knowledge) and $O(n^{2/3})$ search time.
- *k*d-trees don't suffer from the mathematical slowdowns of BSPs because their planes are always axis-aligned.

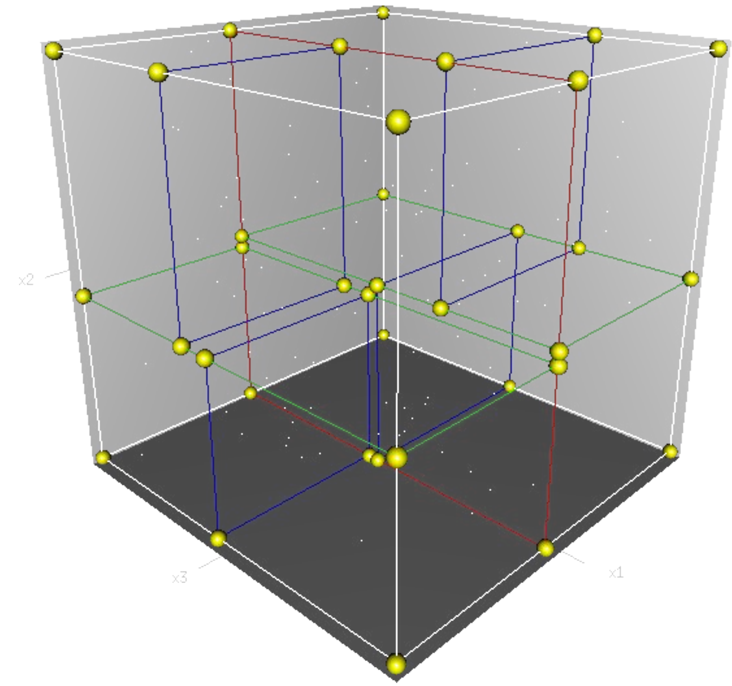


Image from Wikipedia, bless their hearts.

Popular acceleration structures: *Bounding Interval Hierarchies*

The *Bounding Interval Hierarchy* subdivides space around the volumes of objects and shrinks each volume to remove unused space.

- Think of this as a “best-fit” *k*d-tree
- Can be built dynamically as each ray is fired into the scene
- Retains implicit contents sorting, which is nice for traversal

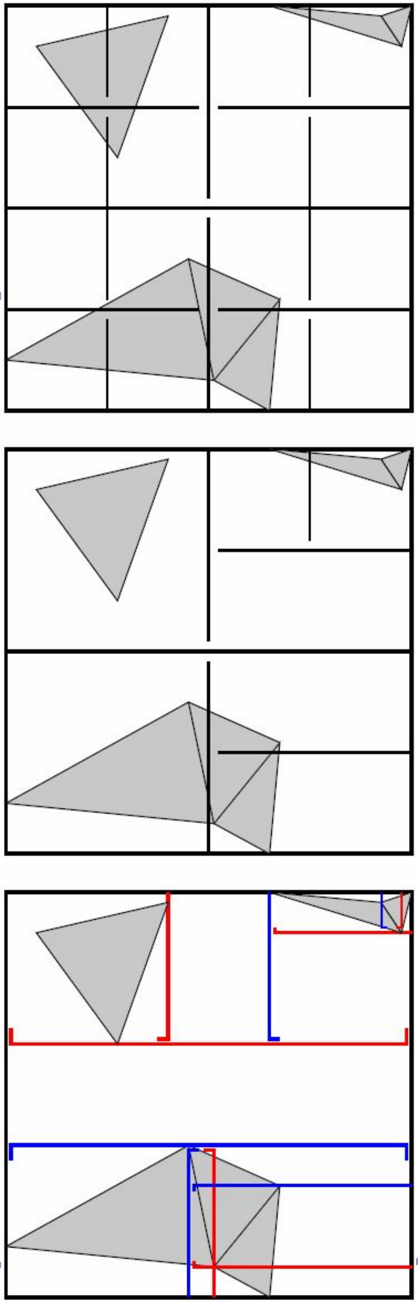


Image from Wächter and Keller’s paper, *Instant Ray Tracing: The Bounding Interval Hierarchy*, Eurographics (2006)

Convex hull

The *convex hull* of a set of points is the unique surface of least area which contains the set.

- If a set of infinite half-planes have a finite non-empty intersection, then the surface of their intersection is a convex polyhedron.
- If a polyhedron is convex then for any two faces A and B in the polyhedron, all points in B which are not in A lie to the same side of the plane containing A.

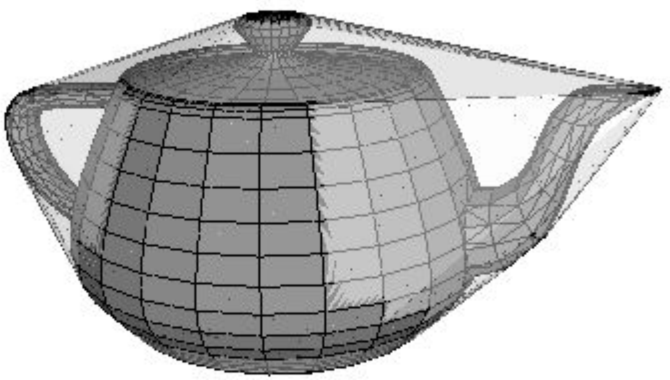
Every point on a convex hull has non-negative angle deficit.

The faces of a convex hull are always convex.

Finding the convex hull of a set of points

Method 1: For every triple of points in the set, define a plane *P*. If all other points in the set lie to the same side of *P* (dot-product test) then add *P* to the hull; else discard.

Problem 1: this works but it's $O(n^4)$.



Finding the convex hull of a set of points

Method 2:

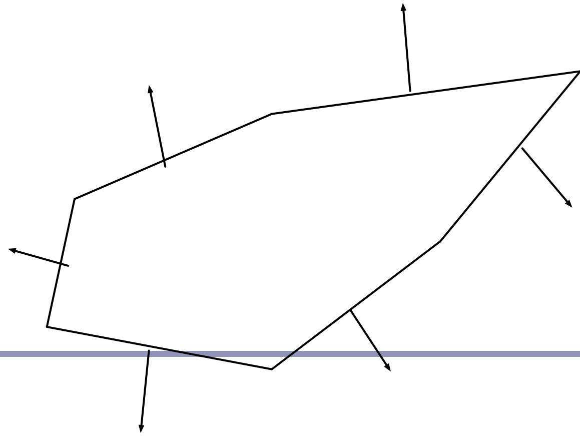
- Initialize *C* with a tetrahedron from any four non-colinear points in the set. Orient the faces of *C* by taking the dot product of the center of each face with the average of the vertices of *C*.
- For each vertex *v*,
  - For each face *f* of *C*,
    - If the dot product of the normal of *f* with the vector from the center of *f* to *v* is positive then *v* is 'above' *f*.
    - If *v* is above *f* then delete *f* and update a (sorted) list of all new border vertices.
  - Create a new triangular face from *v* to each pair of border vertices.

Time complexity: $O(n^2)$

Testing if a point is inside a convex hull

We can generalize Method 2 to test whether a point is inside any convex polyhedron.

- For each face, test the dot product of the normal of the face with a vector from the face to the point. If the dot is ever positive, the point lies outside.
- The same logic applies if you're storing normals at vertices.



References

Voronoi diagrams

- M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, "*Computational Geometry: Algorithms and Applications*", Springer-Verlag,
- http://www.cs.uu.nl/geobook/
- http://www.ics.uci.edu/~eppstein/junkyard/nn.html
- http://www.iquilezles.org/www/articles/voronoilines/voronoilines.htm

Gaussian Curvature

- http://en.wikipedia.org/wiki/Gaussian_curvature
- http://mathworld.wolfram.com/GaussianCurvature.html

The Poincaré Formula

- http://mathworld.wolfram.com/PoincareFormula.html