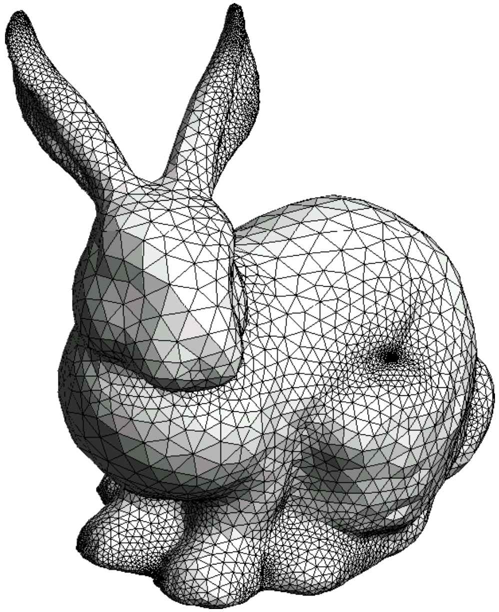


Geometric Modeling in Graphics

Part 2: Meshes properties

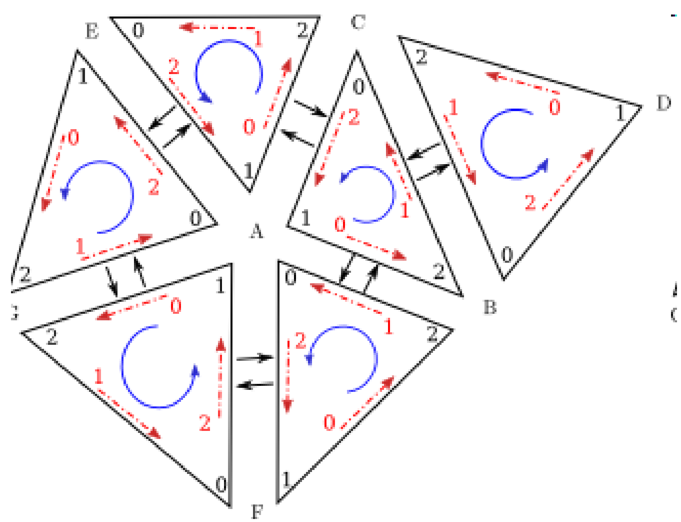
Martin Samuelčík
www.sccg.sk/~samuelcik
samuelcik@sccg.sk

Meshes properties

- Working with DCEL representation
- One connected component with simple polygons
- Basic properties of mesh used in modeling
  - Orientation
  - Area, volume
  - Normal
  - Curvature
  - Interior & exterior
  - Intersections
  - Distances
  - Descriptor & comparison
  - Parametrization
  - Bounding box
  - Skeleton
  - …

DCEL mesh orientation

- Given by order of vertices in faces = order of half edges in faces
- For each half edge, its opposite half edge must have flipped orientation = opposite half edges can not have same origin vertex
- Fixing orientation – making proper orientation in faces, if possible



DCEL mesh orientation fix

```
FixOrientation(DCEL mesh)
{
    List<Face> processed_faces;
    Face current_face = mesh.faces[0];
    while (current_face != null)
    {
        HalfEdge current_edge = current_face.edge;
        do
        {
            int num_flip_edges = 0, num_noflip_edges = 0;
            if (current_edge.opp != null &&
                processed_faces.Contains(current_edge.opp.face))
            {
                if (current_edge.origin == current_edge.opp.origin)
                    num_flip_edges++;
                else
                    num_noflip_edges++;
            }
            current_edge = current_edge.next;
        }
        while (current_edge != current_face.edge)
        if (num_flip_edges > 0 && num_noflip_edges > 0)
            return false;
        if (num_flip_edges > 0)
            FlipOrientation(current_face);
        processed_faces.Add(current_face);
        current_face = GetNextUnprocessedFace(processed_faces);
    }
    return true;
}
```

```
GetNextUnprocessedFace(List<Face> processed_faces)
{
    foreach (Face face in processed_faces)
    {
        HalfEdge current_edge = face.edge;
        do
        {
            if (current_edge.opp != null &&
                !processed_faces.Contains(current_edge.opp.face))
                return current_edge.opp.face;
            current_edge = current_edge.next;
        }
        while (current_edge != face.edge)
    }
    return null;
}
```

```
FlipOrientation(Face face)
{
    HalfEdge current_edge = face.edge;
    HalfEdge prev_edge = null;
    do
    {
        HalfEdge old_next = current_edge.next;
        if (prev_edge != null) current_edge.next = prev_edge;
        current_edge.origin = old_next.origin;
        current_edge.origin.edge = current_edge;
        prev_edge = current_edge;
        current_edge = old_next;
    }
    while (current_edge != face.edge)
    face.edge = prev_edge;
}
```

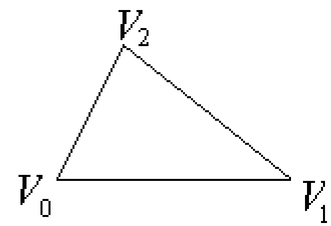
Mesh area

- Mesh area - sum of areas for polygons
- For triangle, (oriented) area A using cross product

$$2\,A(\Delta) = \begin{vmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{vmatrix} = \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

$$= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

where $V_i = (x_i, y_i)$



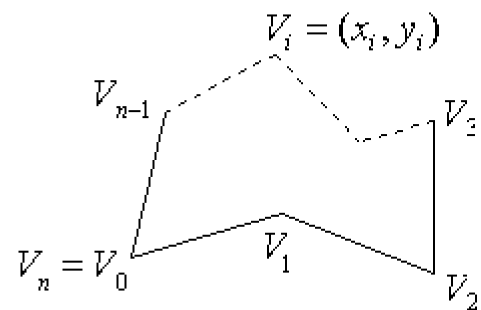
- Oriented area A for simple polygon in 2D

$$2\,A(\Omega) = \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

$$= \sum_{i=0}^{n-1} (x_i + x_{i+1})(y_{i+1} - y_i)$$

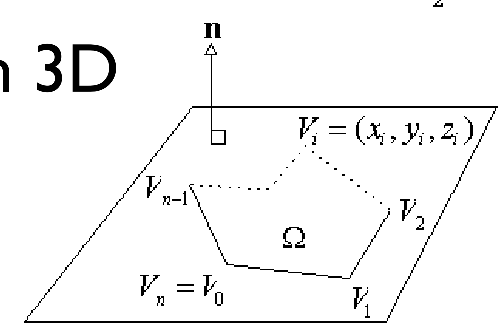
$$= \sum_{i=1}^{n} x_i (y_{i+1} - y_{i-1})$$

where $V_i = (x_i, y_i)$, with $i \pmod n$



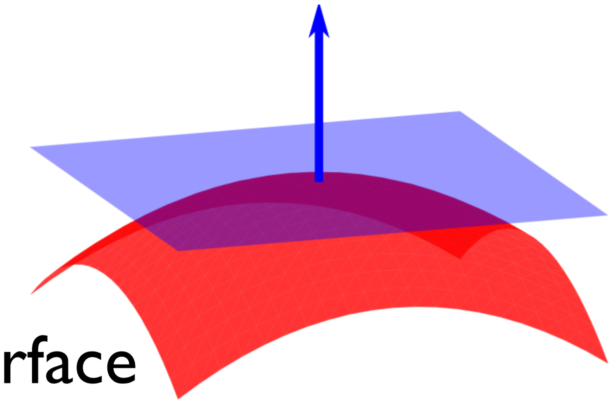
- Oriented area A for simple polygon in 3D

$$2\,A(\Omega) = \mathbf{n} \cdot \sum_{i=0}^{n-1} (V_i \times V_{i+1})$$



http://geomalgorithms.com/a01-_area.html

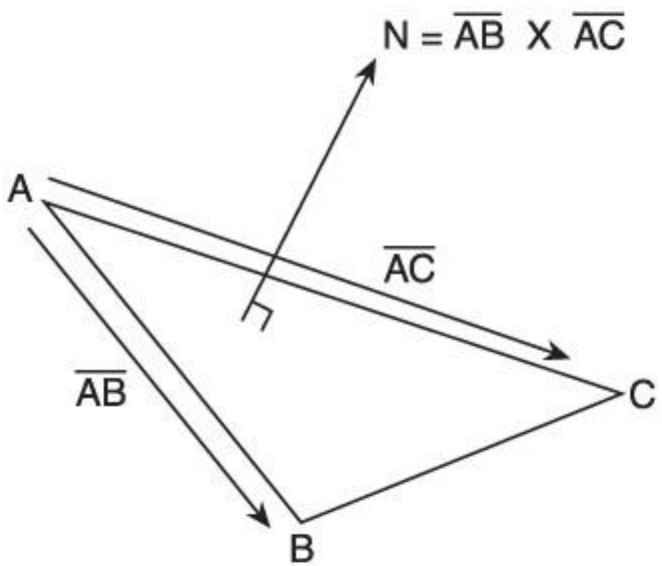
Mesh normals



- Unit vector perpendicular to plane
- Normal of tangent plane of point on surface
- For geometric normal, derivation at point is needed
- Face normal
  - Oriented normal of face plane
  - Direction given by orientation of face
  - Used for determining side of face (face culling, interior, …)
- Vertex pseudo-normal
  - Attribute of vertex
  - No derivation in vertex - normal of some approximation surface passing vertex
  - Used for modeling and visualization (illumination models, …)
  - Not always given by geometric properties

Face normal

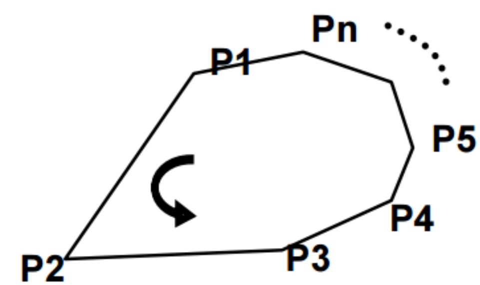
- For triangle, determined by cross product
- If given triangle ABC (in this order), then face normal N is computed as cross product of AB and AC (in this order)
- General face normal N for (nonplanar) polygon (P1,P2,…,Pn)



Pi=[xi, yi, zi], i=1,2,…,n
N=[Nx, Ny, Nz]

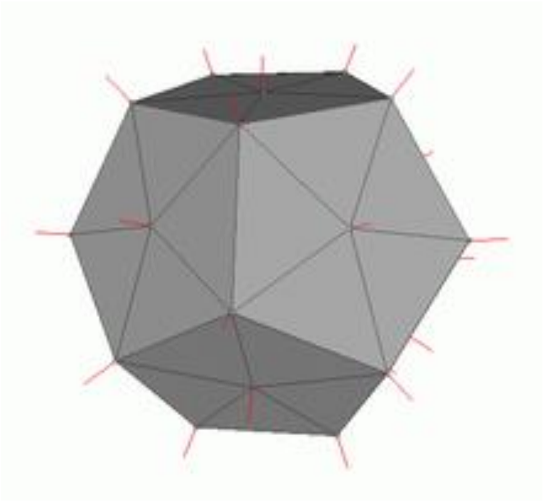
$$N_x = \Sigma\,(y_j - y_i)(z_j + z_i)$$
$$N_y = \Sigma\,(z_j - z_i)(x_j + x_i)$$
$$N_z = \Sigma\,(x_j - x_i)(y_j + y_i)$$

j= (i+1) mod n



Vertex normal

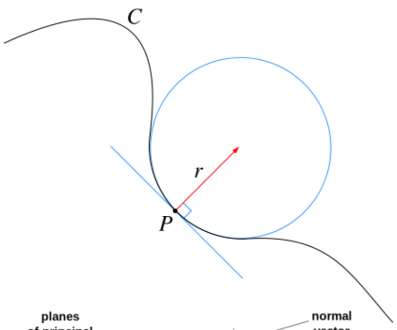
- Usually computed as weighted average of adjacent faces
- Weight of i-th face Fi
  - wi=1
  - wi = Area(Fi)
  - wi = Angle(Fi, v)
  - Weights must be normalized



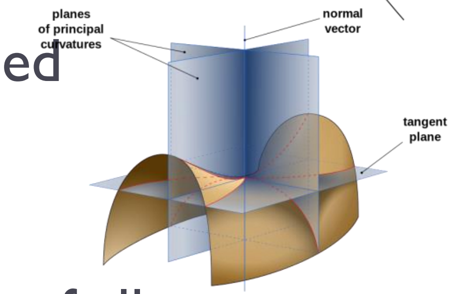
```
ComputeVertexNormalAreaWeights(Vertex v)
{
    Vector N(0, 0, 0);
    float total_weight = 0;
    HalfEdge current_edge = v.edge;
    do
    {
        float wi = FaceArea(current_edge.face);
        total_weight += wi;
        N += wi * ComputeFaceNormal(current_edge.face);
        if (current_edge.opp == null)
            break;
        current_edge = current_edge.opp.next;
    }
    while (current_edge != v.edge);
    current_edge = v.edge.prev.opp;
    do
    {
        if (current_edge == null) break;
        float wi = FaceArea(current_edge.face);
        total_weight += wi;
        N += wi * ComputeFaceNormal(current_edge.face);
        if (current_edge.prev.opp == null)
            break;
        current_edge = current_edge.prev.opp;
    }
    while (current_edge != v.edge);
    return Normalize(N / total_weight);
}
```

Curvature

- How much is curve or surface curved at given point
- Curves
  - Straight line has curvature equal to 0
  - At given point, best possible circle is fitted
  - Curvature is reciprocal of fitted circle radius
  - For computation, second order derivation is needed



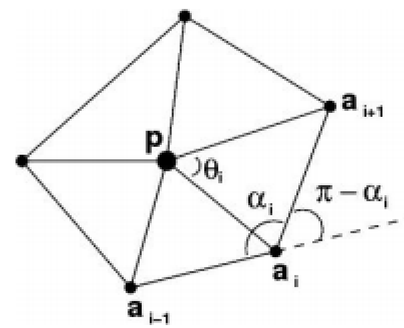
$$k = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}}.$$



- Surfaces
  - At given point, and given tangent vector, curvature of all curves passing that point with that tangent vector is the same
  - There is maximum and minimum of all tangent curvatures – principal curvatures k1, k2
  - Gaussian curvature K = k1.k2, mean curvature H = 0.5*(k1+k2)

Mesh curvature

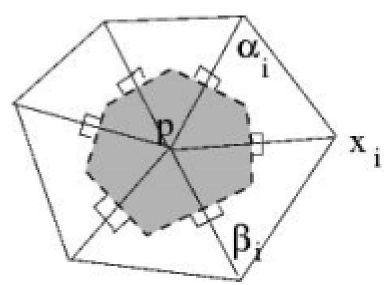
- Polygonal esh – no first and second order derivation on edges and at vertices
- Curvature equal to 0 inside faces
- „Curvature“ at vertex – curvature of some interpolation surface at vertex
- Gaussian curvature for triangle meshes



$$K_g = \frac{1}{\mathcal{A}}\left(2\pi - \sum_{j=1}^{N} \Theta_j\right)$$

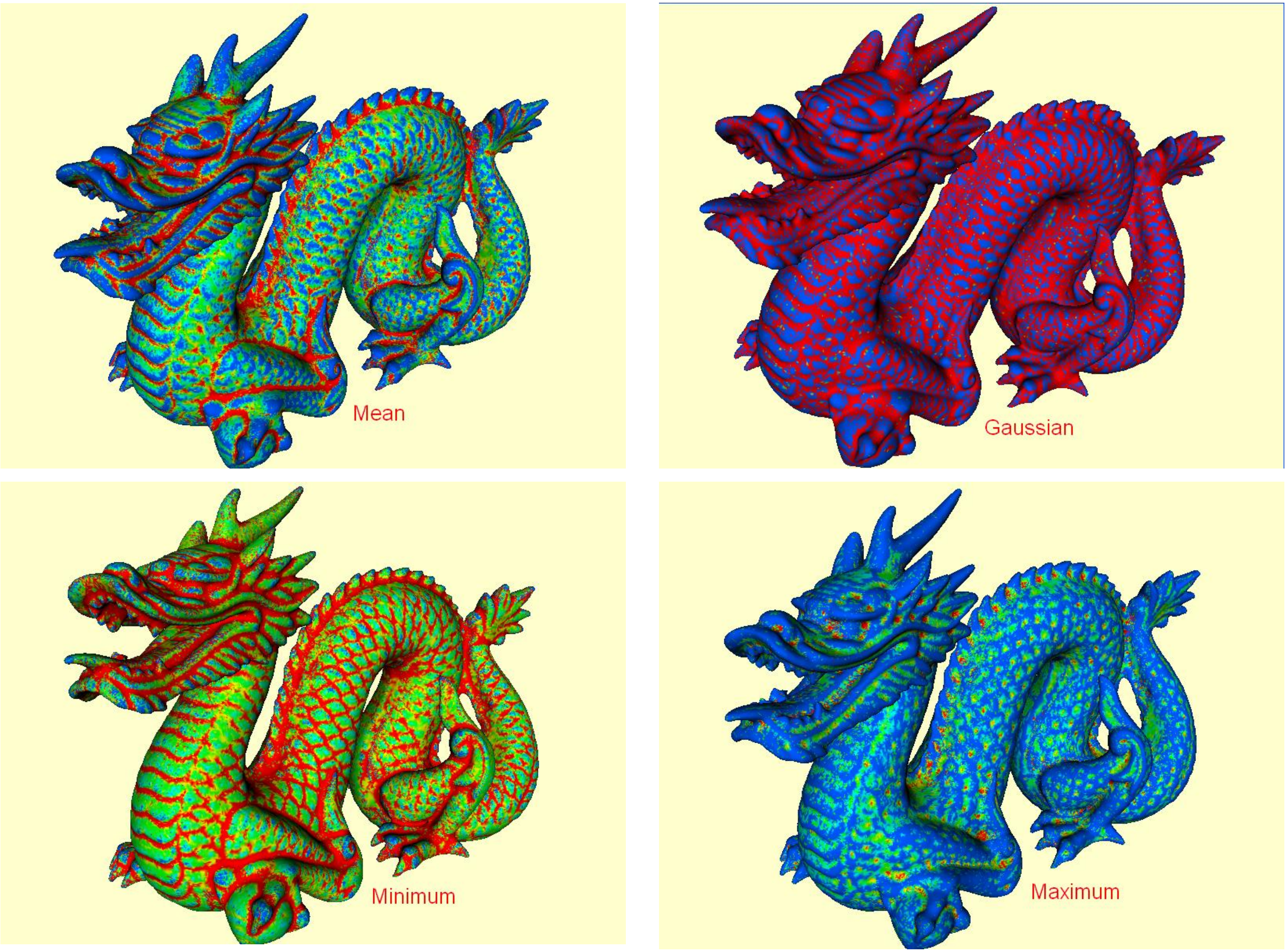
- Mean curvature for triangle meshes

$$H_p = \frac{1}{4\mathcal{A}} \| \sum_{i \in st(p)} (\cot\alpha_i + \cot\beta_i)(x_i - p) \|.$$



ftp://ftp.disi.unige.it/person/MagilloP/PDF/lncs2012.pdf

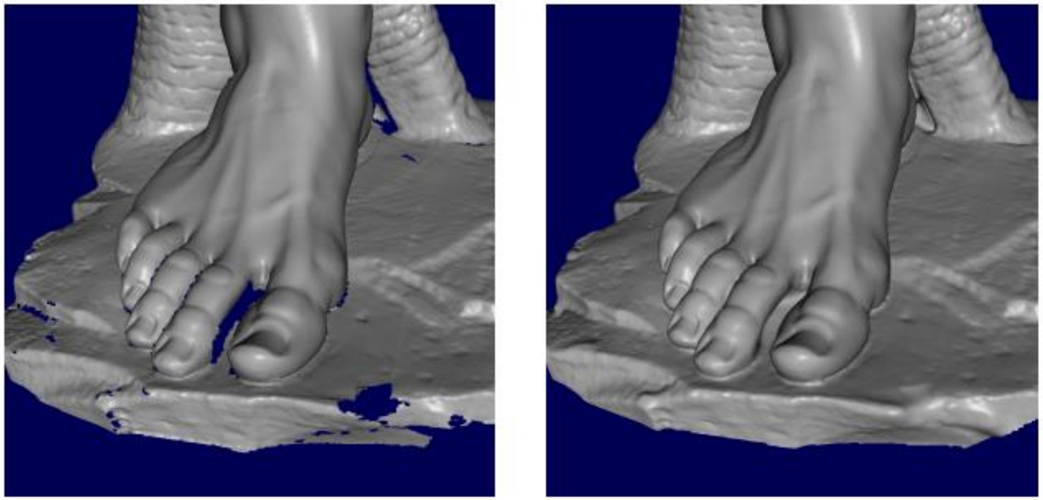
Mesh curvatures



http://graphics.ucsd.edu/~iman/Curvature/

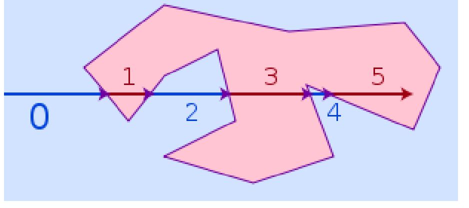
Closed mesh

- Mesh dividing space to two sets, interior and exterior
- Interior and exterior should be non-empty sets
- Unclosed mesh has some holes, and has some boundary edges – edges with only one adjacent face
- Mesh in DCEL representation is closed if all opposite pointers in all half edges are non-null

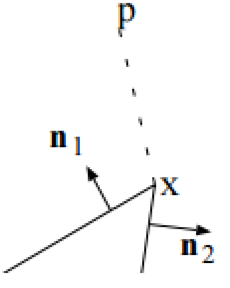


Interior determination

- Check if given point in interior or exterior set of mesh
- 1. Cast ray from point, if it hits mesh in odd number if intersections, it is inside mesh, and outside otherwise

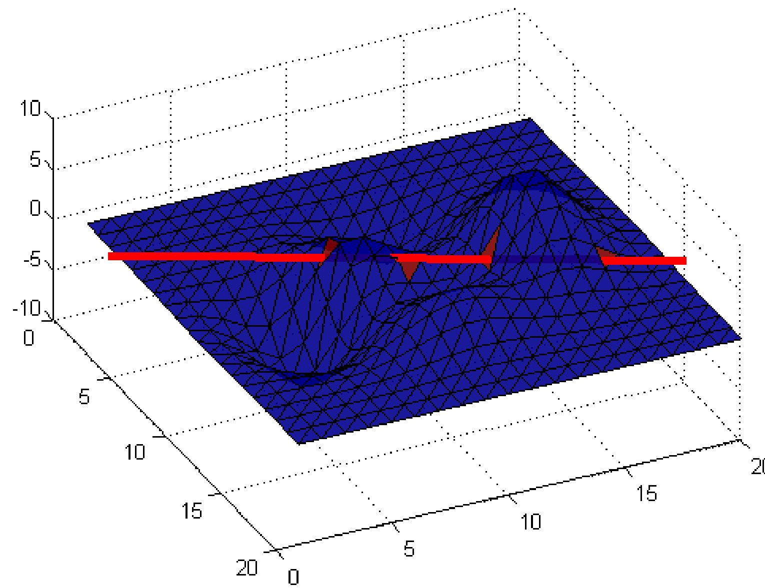


- 2. Find closest point C of given point P on mesh, then use dot product of P-C and normal in C to determine if it is inside or outside. Use angle-weighted pseudo normal if C is vertex or on edge of mesh.



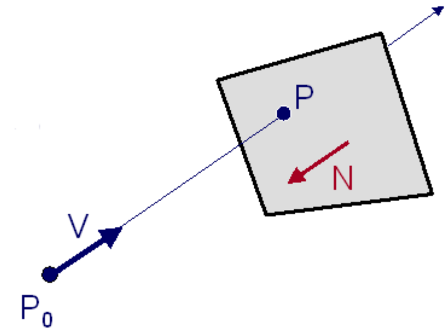
Ray-mesh intersections

- Finding intersections of ray and polygons of mesh
- Counting intersections on edges and in vertices only once
- Usually checking for intersection of ray and triangle
- Using acceleration structures
  - Uniform grid
  - Octree
  - kd-tree
  - Bounding volumes hierarchy



Ray-triangle intersection

- Find intersection of ray and plane
  - Ray: $P=P_0+tV$
  - Plane: $P.N+d=0$
  - $t=-(P_0.N+d)/(V.N)$

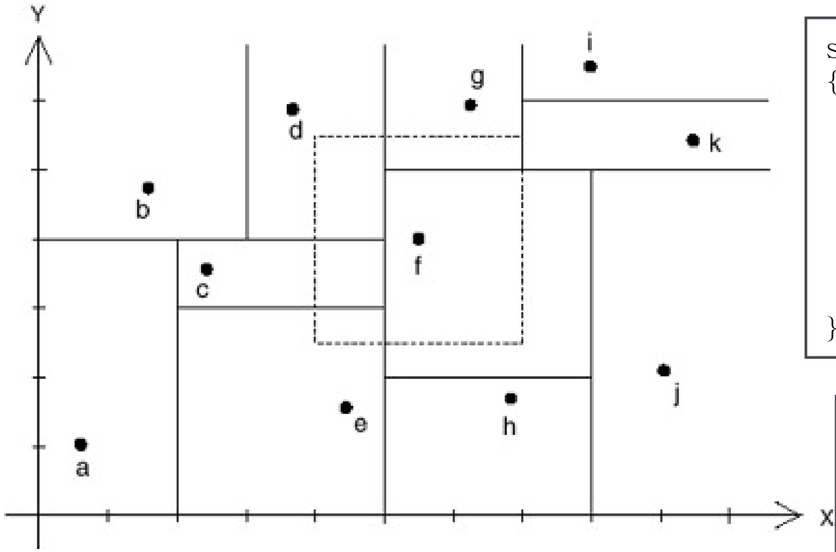


- Find if intersection point lies inside triangle
  - A,B,C – coordinates of triangle vertices
  - P=uA+vB+wC, u+v+w=1, barycentric coordinates
  - Three equations, three variables u,v,w
  - If 0 <= u,v,w ,= 1, then P is inside ABC
- Optimized computations
  - https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm

Kd-tree

- Probably fastest supporting structure for ray-mesh intersection
  - http://dcgi.felk.cvut.cz/home/havran/phdthesis.html
- Binary tree structure, each node containing one dividing plane perpendicular to one coordinate axis – each node represents axis-aligned convex area of space
- Polygons of mesh are stored only in leafs
- All polygons stored in subtree of a node are inside of the node area
- When finding intersections of ray and mesh, first kd-tree is traversed and only nodes intersecting with ray are visited
- Ray-polygon intersections are computed only for visited leafs
- Used also for set of meshes

Kd-tree

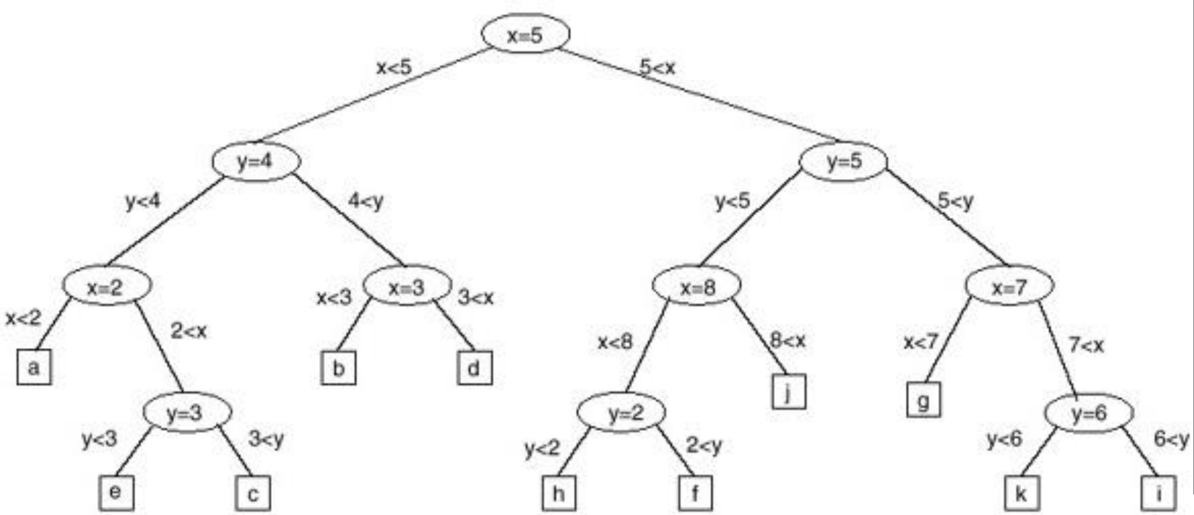



```
struct KdTreeNode
{
    float  split;
    int dim;
    List<Face> data;
    KdTreeNode * left;
    KdTreeNode * right;
    KdTreeNode * parent;
}
```

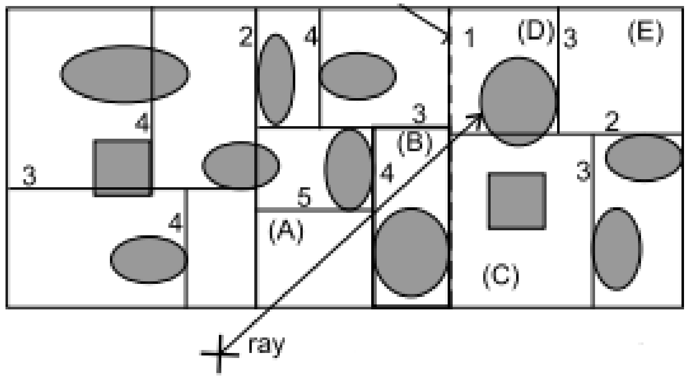
```
struct KdTree
{
    KdTreeNode * root;
}
```

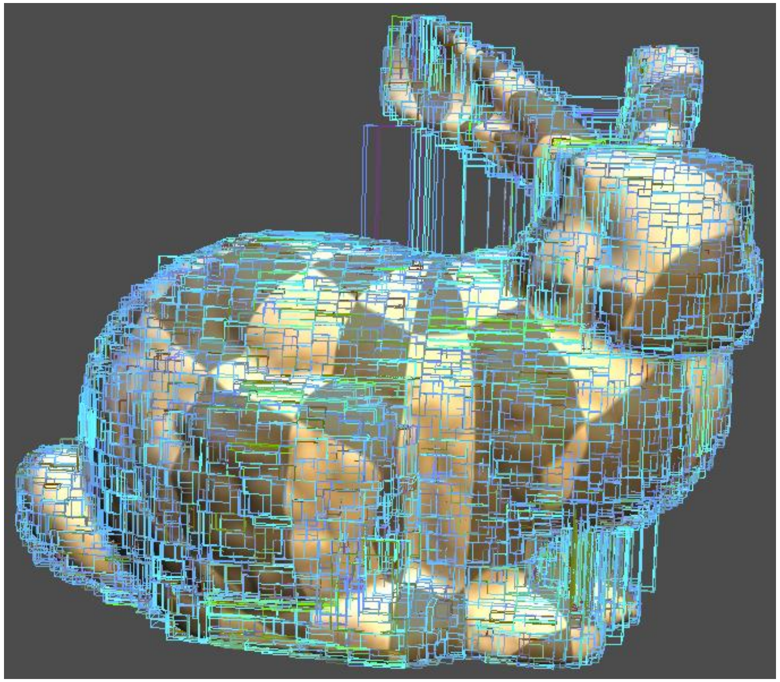
```
KdTreeConstruct(S, d)
{
    T = new KdTree;
    T->root = KdTreeNodeConstruct(S, 0, d);
    return T;
}
```

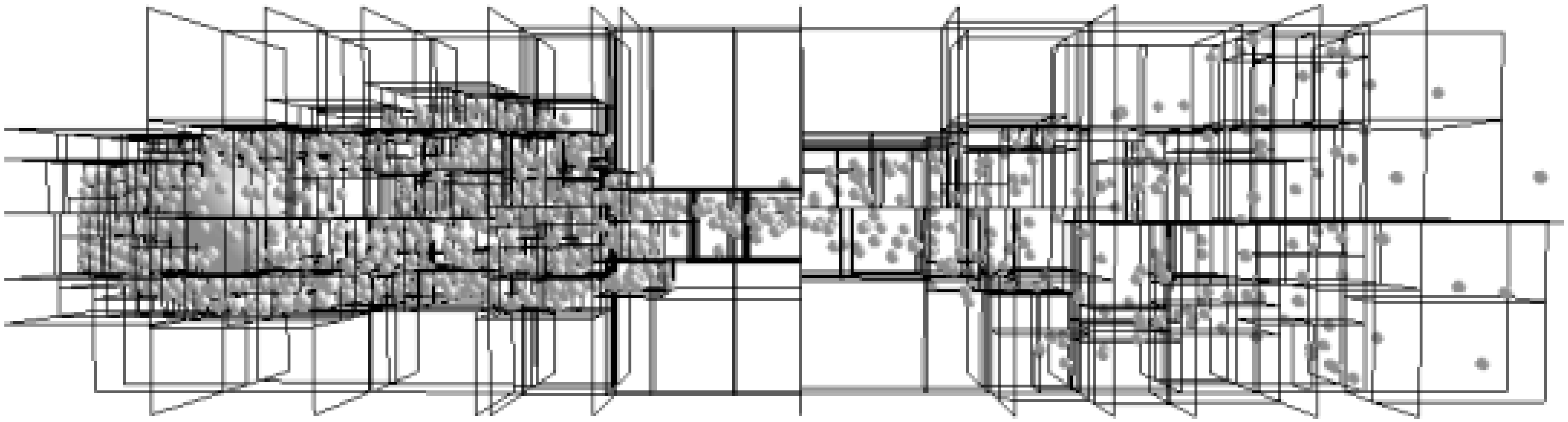
```
KdTreeNodeConstruct(D, dim, d)
{
    if (|D| = 0)  return null;
    v = new KdTreeNode;
    v->dim = dim;
    if (|D| <= THRESHOLD)
    {
        v->data = D.Elements;
        v->left = null;
        v->right = null;
        return v;
    }
    v->data = null;
    v->split = D.ComputeSplitValue(dim);
    D<s = D.Left(dim, v->split);
    D>s = D.Right(dim, v->split);
    j = (dim +  1) mod d;
    v->left = KdTreeNodeConstruct(D<s, j);
    v->right = KdTreeNodeConstruct(D>s, j);
    return v;
}
```



Kd-tree





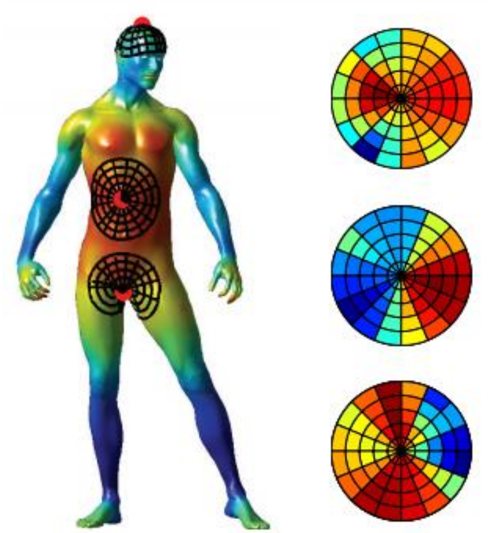


Mesh descriptors

- Describing mesh using small number of numbers – descriptor vector
- If description vectors are same, then meshes should be same and vice versa
- Similar meshes has similar vector using some vectors comparison metrics
- Used for mesh comparisons, shape recognition, shape retrieval, …
- Transformation invariance
- http://web.ist.utl.pt/alfredo.ferreira/publications/DecorAR-Surveyon3DShapedescriptors.pdf

Shape Contexts

- Divide space into smaller number of bins, centered at local point or global center
- Prepare normalized histogram for number of mesh vertices inside bins
- Global
  - Uniform grid over whole mesh
  - Count number of vertices for each cell (bin)
  - Normalized count is descriptor vector
- Local
  - Put disc grid at each vertex location and count number of vertices in local neighborhood

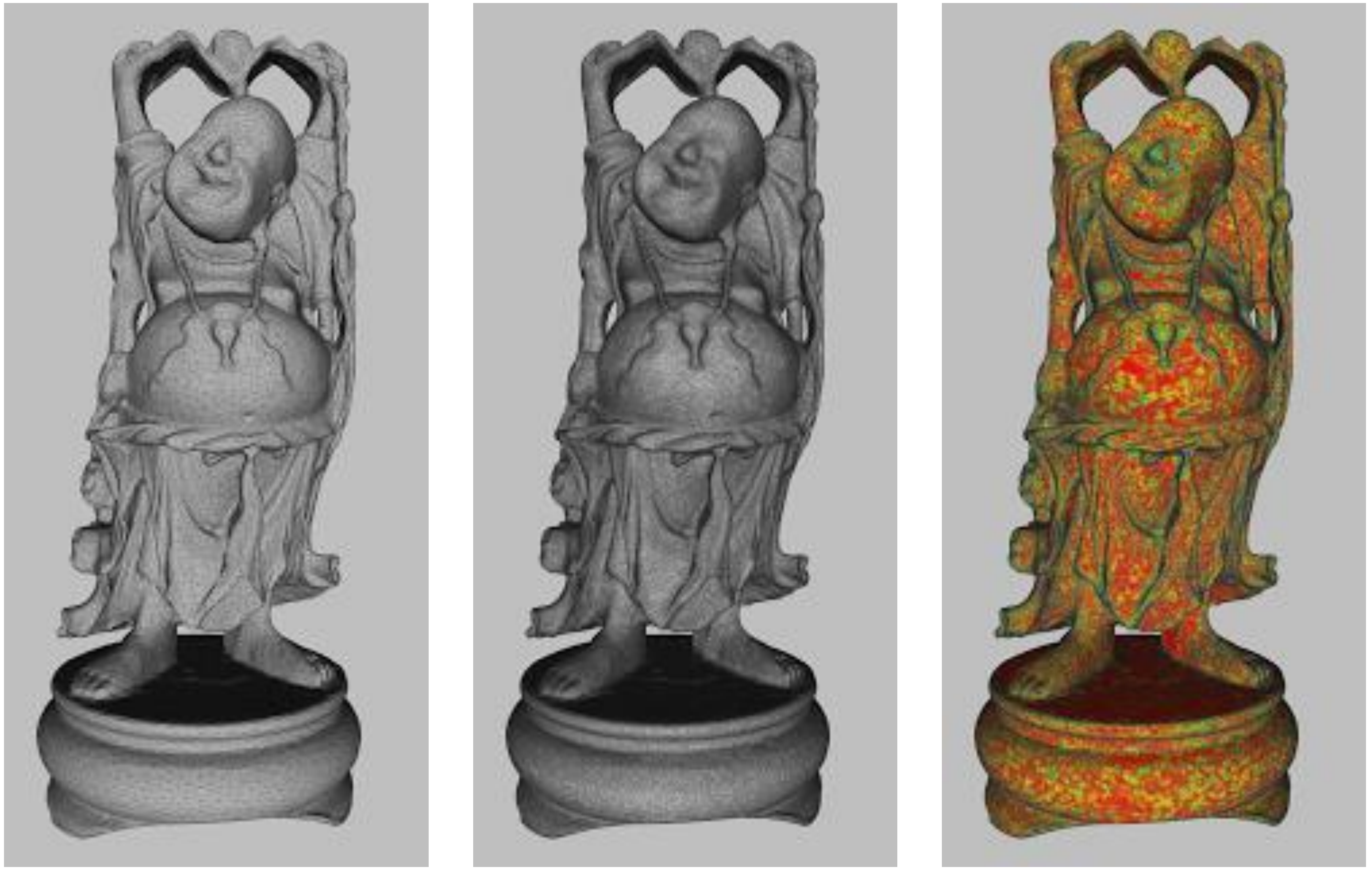


Hausdorff distance

- Point-mesh distance (point x, mesh A)
  - d(x,A) = inf{d(x,y);y in A};
- Mesh-mesh Hausdorff distance (mesh A, mesh B)
  - d(A,B) = sup{d(x,B);x in A}
- Symmetrical mesh-mesh Hausdorff distance (mesh A, mesh B)
  - h(A,B) = max{d(A,B),d(B,A)}
- If 0, meshes are identical
- Higher distance = meshes are more different
- For computation, acceleration structures like uniform grid, octree, kd-tree are used
- http://www.cmap.polytechnique.fr/~peyre/cours/x2005signal/mesh_mesh.pdf

Hausdorff distance

- http://meshlabstuff.blogspot.sk/2010/01/measuring-difference-between-two-meshes.html



Mesh bounding box

- Finding tight bounding box for mesh and principal direction
- Using PCA (Principal component analysis)
- Using vertices of mesh $V_i=[x_i,y_i,z_i]$
- http://jamesgregson.blogspot.sk/2011/03/latex-test.html
- 1. Compute mean position for each coordinate

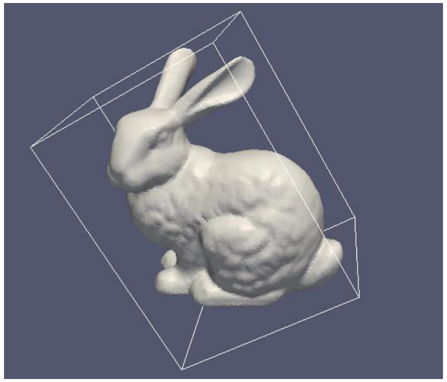
$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \hat{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \qquad \hat{z} = \frac{1}{N}\sum_{i=1}^{N} z_i$$

$$\mathbf{C} = \begin{bmatrix} E[xx]-\hat{x}\hat{x} & E[xy]-\hat{x}\hat{y} & E[xz]-\hat{x}\hat{z} \\ E[yx]-\hat{y}\hat{x} & E[yy]-\hat{y}\hat{y} & E[yz]-\hat{y}\hat{z} \\ E[zx]-\hat{z}\hat{x} & E[zy]-\hat{z}\hat{y} & E[zz]-\hat{z}\hat{z} \end{bmatrix}$$

- 2. Compute covariance matrix C

$$E[xx] = \frac{1}{N}\sum_{i=1}^{N} x_i x_i \qquad E[xy] = \frac{1}{N}\sum_{i=1}^{N} x_i y_i, \qquad \ldots, \qquad E[xz] = \frac{1}{N}\sum_{i=1}^{N} x_i z_i$$

- 3. Find eigenvectors of covariance matrix C
- 4. Eigenvectors form orthogonal frame of oriented bounding box



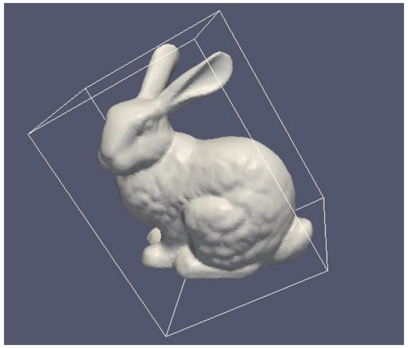
Mesh bounding box

- Using triangles instead of vertices, A_i is are of i-th triangle, A_m is area of entire mesh, p,q,r are vertices of i-th triangle

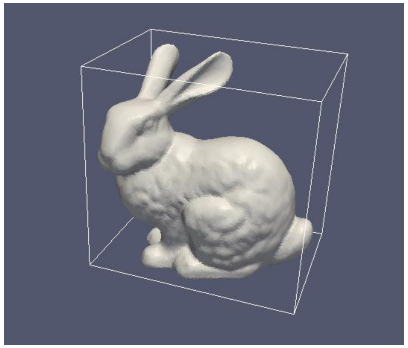
$$\hat{x} = \frac{1}{A_m}\sum_{i=1}^{N} A_i \hat{x}_i \qquad \hat{y} = \frac{1}{A_m}\sum_{i=1}^{N} A_i \hat{y}_i \qquad \hat{z} = \frac{1}{A_m}\sum_{i=1}^{N} A_i \hat{z}_i$$

$$E[xx] = \frac{1}{A_m}\sum_{i=1}^{N} \frac{A_i}{12}(9\hat{x}_i\hat{x}_i + p_x p_x + q_x q_x + r_x r_x) \qquad E[xy] = \frac{1}{A_m}\sum_{i=1}^{N} \frac{A_i}{12}(9\hat{x}_i\hat{y}_i + p_x p_y + q_x q_y + r_x r_y)$$

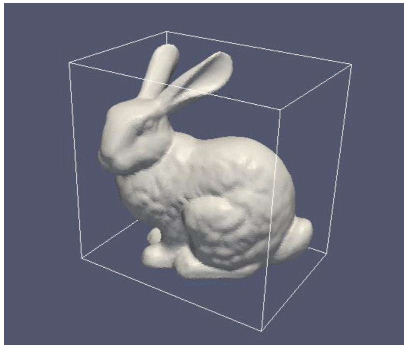
- Using only vertices or triangles from convex hull of mesh
- Using only one eigenvector from PCA, other 2 directions are computed using 2D PCA from projected vertices



OBB fit using points



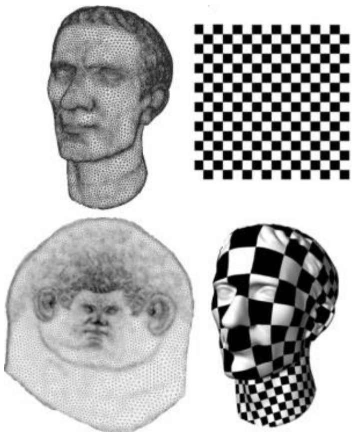
OBB fit using triangles



OBB fit using convex hull

Mesh parameterization

- Polygonal mesh – 2D object, manifold
- Parameterization – finding bijective mapping of 2D plane and polygonal mesh
- Usually defined by putting 2 coordinates (u,v) at each vertex – defining coordinates of vertex in 2D space
- 2D coordinates of points inside faces are computed using interpolation
- https://igl.ethz.ch/teaching/tau/adv_cg/Parameterization03_I.ppt
- Usage
  - Texture mapping
  - Mesh editing
  - Morphing
  - Animation

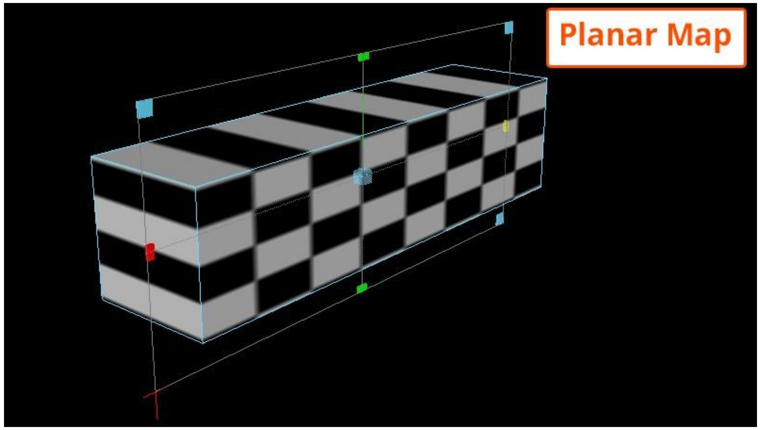


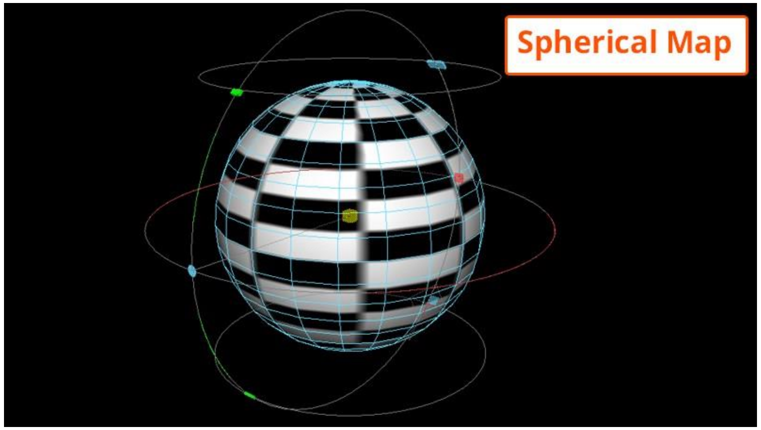
Basic parameterizations

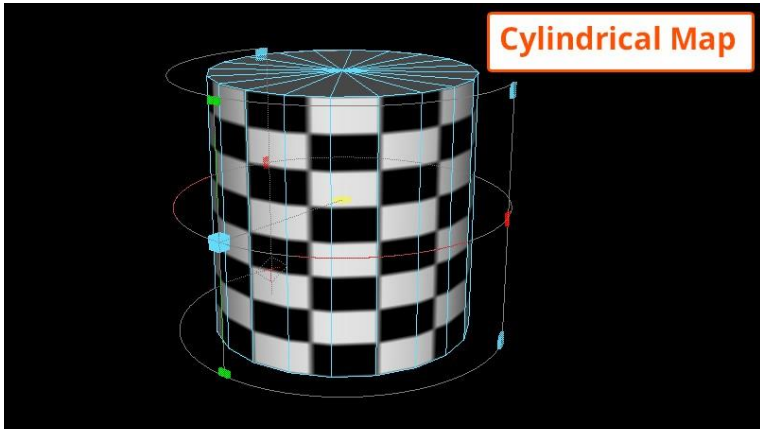
- Computing u,v for each vertex V_i
- Planar
  - Given plane by origin O and two orthonormal vector X,Y
  - $u = (Vi-O)\cdot X$, $v=(Vi-O)\cdot Y$
- Spherical
  - Given origin O
  - $r=|V_i-O|$, $u=atan(V_{ix}-O_x)/(V_{iy}-O_y))$, $v=acos((V_{iz}-O_z)/r)$
- Cylindrical
  - Given origin O
  - $R=sqrt((V_{iz}-O_x)^2+(V_{iy}-O_y)^2)$, $u=asin((V_{iy}-O_y)/r)$,$v=V_{iz}-O_z$

Basic parameterizations

- http://blog.digitaltutors.com/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know/







The End
for today