1 Mesh related topics

1.1 Level of Detail (LOD)

We use the term *resolution* to refer to the size of polygons in the model, for example, the length of the shortest edge. As we discussed last class in the context of fractals, if we limit ourselves to a given smallest edge size (say 1 mm) then there may be details in the surfaces geometry at a smaller size that we cannot capture.

The resolution at which an object should be displayed in an image might be much coarser then the resolution of the model in the database. For example, if a mudpile or rockface were viewed from up close then it would require a large percentage of the pixels in the image and a fine details of the model would be needed. But if the surface were seen from a great distance, then it would occupy only a relative small number of pixels in the image and only a coarse resolution would be needed. One often refers to the *level of detail* (LOD) of a model.[1]

One approach to having multiple levels of detail is to *precompute* several models that differ in resolution. For example, a low resolution model might have one 10^2 polygons, a medium resolution model might have 10^4 polygons, and a high resolution model might have 10^6 polygons, etc.

How do you choose which model to use at any time? The basic idea is that when drawing an object, you might first calculate roughly how many pixels the object would occupy in the image (using a bounding box for the object, say) and then decide which resolution model to use based on this number of pixels.

For example, suppose you wanted to use the Stanford model of Michelangelo's David in an image, such that David is viewed from a distance of 20 meters and occupies only a small fraction of the image. Obviously it would be wasteful to process every one of the two billion polygons in the model. Instead, we would use a lower resolution model.

A second approach is to represent the model using a data structure that allows you to vary the number of polygons by simplifying the polygonal mesh in a near-continuous way (see below). For example, you could let the resolution of the model vary across the object in a *viewpoint dependent* way. In an outdoor scene, the part of the ground that is close to the camera would need a higher LOD, whereas for the part that is further from the camera a lower LOD would suffice. Thus, we could use a coarse resolution for distant parts of the scene and a fine resolution for nearby parts of the scene. For the example of the terrain on page 2, the surfaces in the upper part of the image which are way off in the distance do not need to be represented as finely (in world coordinates i.e. $\Re^3$) as the polygons at the bottom of the image which are relatively close to the camera.

1.2 Mesh Simplification

How do we obtain models with different level of detail ? For example, suppose we are given or we have constructed a triangular mesh having a large number of triangles. From this high LOD mesh, we would like to compute a lower LOD mesh. This process is called *mesh simplification*. Let's discuss one approach.

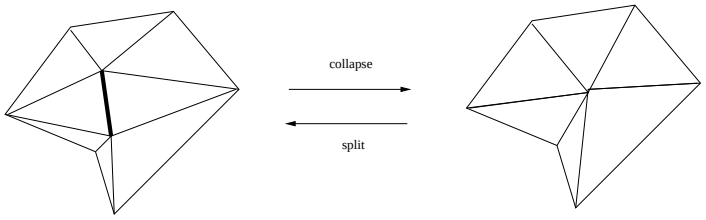
1.2.1 Edge collapse method

Suppose we are given a triangulated mesh with N vertices, and we wish to simpify it to mesh having $N-1$ vertices. As shown in the example below, we can collapse one of the edges (defined by a pair of vertices) to a single vertex.

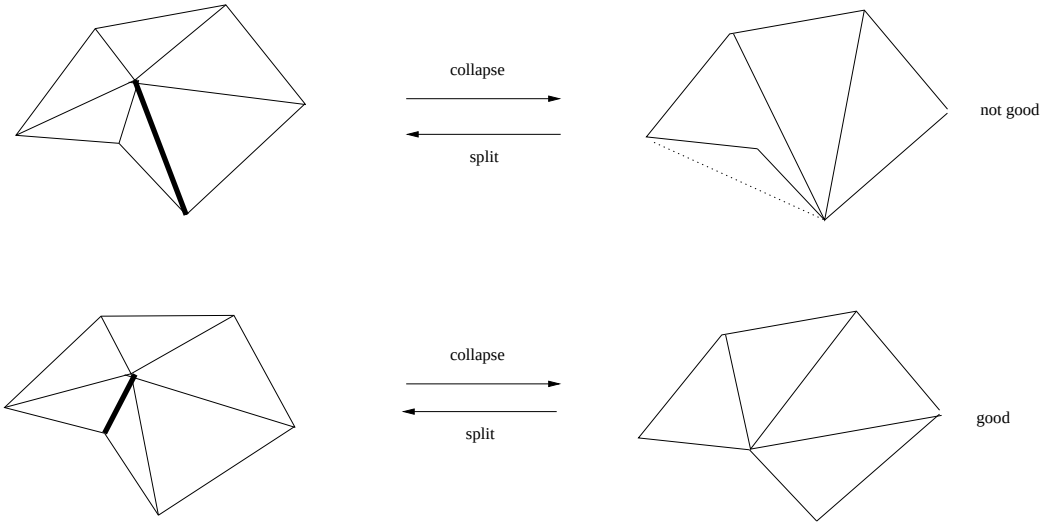
[1] See nice book "Level of Detail for 3D Computer Graphics" by David Luebke *et al.* 2003.

There are two ways to do this. One way is to delete one of the vertices of the collapsed edge. The other is to replace both vertices of the edge by one new vertex. Either way, a new triangulation must be computed to take account of the fact that vertices have been deleted or replaced.

Notice that when you do an edge collapse as shown in the figure below, you lose two triangles, one vertex, and three edges. You can repeat this and decrease N to be as small as you like.



Subtle issues can arise, for example, faces can be flipped. In the second example below, the boldface edge is collapsed and all edges that are connected to the upper vertex are deleted and/or replaced by an edge to the lower vertex. Consider one such new edge, namely the dotted one shown on the right. This dotted edge in fact belongs to *two* triangles: one whose normal points out of the page, and a second whose normal points into the page. The latter is thus flipped. This is not good. For example, if the N vertex surface were a terrain then the $N-1$ vertex surface should also be a terrain (intuitively).

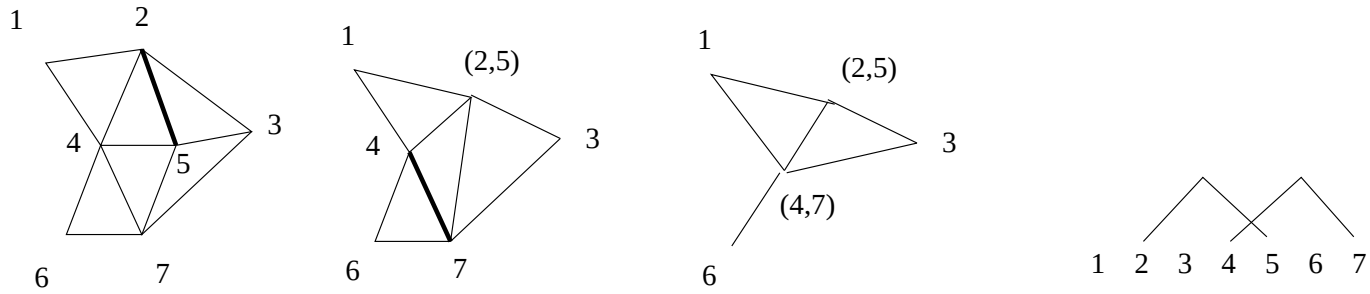


You can organize edge collapses using a binary tree. The leaves of the tree are the vertices of the original mesh. Each edge collapse creates a sibling relationship between two nodes in the tree, such that the parent of these siblings is the vertex that the pair of vertices collapses to.

Start the entire high LOD model, such that each vertex is its own tree (a trivial tree consisting of a root only). Then, if two vertices are connected by an edge and you choose to collapse this edge (based on some error metric, see below), then a parent node is defined for these two vertices and the trees for these two vertices are merged. The representative vertex for the new parent node is either one of its children vertices, or some new vertex that replaces the two children vertices. This information must be stored at the parent vertex, so that it is possible later to *split* the parent vertex, that is, to recover the children vertices (and the edge between them).

As many edges can be collapsed as desired, i.e. one can simplify the mesh from N vertices down to any M vertices, where $M < N$.

Such a data structure can be used to choose the level of detail at run time. One can begin with the simplified mesh, and then expand the mesh by "splitting" (or subdividing) the vertex if the resolution of some part of the mesh is too low based on some criterion – see the discussion of the outdoor scene on the previous page. Note that splitting is the inverse of collapsing (see above fig).



The above example shows the basic idea. Two edge collapses are shown. Although the rightmost "triangulation" doesn't appear to be a triangulation, you must keep in mind that only part of the mesh is shown. In particular, verticles 1, 3, 6 are each connected to other vertices not shown here.

Note that we often want to store additional information with a mesh, such as normal and texture information. So, when we have a binary tree that allows us to represent mesh simplification, we need to store other information at the internal nodes of the tree as well, namely how the values of these attributes on the simplified mesh.

1.2.2 Quadric Error Metric

In the previous section, we discussed a process for collapsing an edge to a vertex to simplify a mesh (and likewise the reverse operation of splitting a vertex to undo this operation). In mesh simplification, the question is thus which edge should be collapsed so that the overall change in shape (i.e., appearance) is minimized.

The quadric error metric is based on the idea that a vertex is naturally found at the intersection of the planes of its adjacent faces. If the vertex is to be moved to another location (i.e., merged with another vertex in order to collapse an edge), then it should try to remain in all of the planes of its surrounding faces. This may sound odd, in the sense that we might want to just keep the vertex close to its initial position; however, if all the adjacent planes are the same, then the vertex has the freedom to move in this plane without changing the shape of the mesh.

Let us now consider how this error metric is formulated. We can write a plane a dot product in homogeneous coordinates,

$$\begin{pmatrix} A & B & C & D \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = 0$$

or simply $p^T v$, where $p = [A\ B\ C\ D]$ gives the coefficients of the plane equation, and v is the position of the vertex. Note that if we choose the plane equation such that $A^2 + B^2 + C^2 = 1$, then the distance

between a vertex v and this plane is given by $p^T v$. The distance squared will always positive, and makes a better measure of error, which we can compute as $(p^T v)^2$, or, $v^T pp^T v$. So, we can define a 4 by 4 matrix $K_p = pp^T$ such that the squared distance between a vertex and the plane is $v^T K_p v$.

Now, recall that a vertex is adjacent to many faces, so we can now write a quadratic error formula for moving this vertex based on all of the adjacent faces. For vertex i, with faces defining a set of planes, planes_i, we can write the error as

$$\begin{aligned} e_i(v) &= \sum_{p \in \text{planes}_i} v^T K_p v \\ &= v^T \left(\sum_{p \in \text{planes}_i} K_p \right) v \\ &= v^T Q_i v \end{aligned}$$

where Q_i is a 4 by 4 matrix which is simply the sum of the matrices K_p for the planes adjacent to vertex i. So, when we collapse an edge between vertices i and j, we want to choose a new vertex position which minimizes the quadric error of both vertices. There are many specific choices, and a few details that are being omitted here, but basically we want to choose a vertex v which minimizes $v^T(Q_i + Q_j)v$. Note that this could run into problems if all the faces lie in the same plane as this does not constrain the position v, but we probably still want the vertex to lie somewhere near the edge between the two vertices.

Up to now, we've been talking about this problem as if the edge was already chosen, but the problem of selecting which edge to collapse can be seen to be related. We can compute the optimal position v for all possible edge collapses, and then choose the collapse that results in the smallest error. More information can be found in Garland and Heckbert's 1997 SIGGRAPH paper (also of interest will be Hoppe's work on Progressive Meshes).

2 Quadric surfaces

A general formula for a quadric surface in $\Re^3$ is

$$ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0.$$

Simple examples are:

- a sphere centered at the origin, $x^2 + y^2 + z^2 = r^2$
- a sphere centered at (x_0, y_0, z_0), namely

  $$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0$$

- an ellipsoid centered at (x_0, y_0, z_0), namely

  $$a(x - x_0)^2 + b(y - y_0)^2 + c(z - z_0)^2 - r^2 = 0$$

  where $a, b, c > 0$
- a cone with apex at (x_0, y_0, z_0), and axis parallel to x axis

  $$a(x - x_0)^2 = (y - y_0)^2 + (z - z_0)^2$$

- a paraboloid with axis parallel to x axis

$$ax = (y-y_0)^2 + (z-z_0)^2$$

- etc

The above general formula for a quadric can be rewritten as:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & \frac{d}{2} & \frac{f}{2} & \frac{g}{2} \\ \frac{d}{2} & b & \frac{e}{2} & \frac{h}{2} \\ \frac{f}{2} & \frac{e}{2} & c & \frac{i}{2} \\ \frac{g}{2} & \frac{h}{2} & \frac{i}{2} & j \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

or

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \mathbf{Q} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

where $\mathbf{Q}$ is a symmetric 4×4 matrix.

Note that we can do ray casting with quadric surfaces, just as we did with polygons. We substitute the three parametric equations of the ray $\{(x(t), y(t), z(t)) : t > 0\}$ into the equation for the quadric surface. This yields a second order "quadratic equation" in the variable t, namely

$$\alpha t^2 + \beta t + \gamma = 0.$$

This equation yields two solutions for t, which may be either real or complex. If this equation has no real solution, then the ray does not intersect the quadric. If the equation has one positive[2] (real) solution, then this solution defines the point on the quadric that is visible along the ray. If the equation has two positive real roots, then the smaller one is used. For example, the quadric might be a sphere, and the ray might pass twice through the sphere – once on the way in, and once on the way out. We choose the first solution since we are interested in the "front face".

Another important note is that if we have a quadric surface in $\Re^3$ then we can scale, translate, rotate, and even apply a (invertible) perspective projection to the quadric and we will get another quadric! To see this, we manipulate the equation

$$\mathbf{x}^T \mathbf{Q} \mathbf{x} = 0$$

where $\mathbf{x} = (x, y, t, 1)^T$. [Notation: Since $(\mathbf{M}^{-1})^T = (\mathbf{M}^T)^{-1}$ for any invertible matrix $\mathbf{M}$, we can simplify the notation by writing just writing either of these as $\mathbf{M}^{-T}$.]

$$\mathbf{x}^T \mathbf{Q} \mathbf{x} = \mathbf{x}^T \mathbf{M}^T \mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1} \mathbf{M} \mathbf{x} = (\mathbf{M}\mathbf{x})^T (\mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1})(\mathbf{M}\mathbf{x})$$

You can easily verify that $\mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1}$ is symmetric, i.e. that

$$\mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1} = (\mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1})^T$$

and so this matrix is a coefficient matrix for a quadric surface. Thus, points $\mathbf{x}$ on the original quadric are transformed to points $\mathbf{M}\mathbf{x}$ on a quadric surface in the projection space, and this quadric is defined by coefficient matrix $\mathbf{M}^{-T} \mathbf{Q} \mathbf{M}^{-1}$.

[2] If the solution is negative, then this point does not lie on the ray (but lies on the line containing the ray).

2.1 Ray Quadric Intersection and Normals

The concept of ray casting was introduced in a previous section as a means of finding what objects contained in a scene are visible to the camera. If the scene contains quadrics, then this ray quadric intersection test reduces to solving a quadratic equation. Consider a ray, $r(t) = r_0 + vt$, which has as origin the point r_0 and is in the direction of vector v. Given a quadric Q, we saw in the previous section that we can find the intersection by solving for the roots of $r(t)^T Q r(t)$, or

$$\begin{aligned}(r_0 + vt)^T Q(r_0 + vt) &= r_0^T Q r_0 + r_0^T Q vt + (vt)^T Q r_0 + (vt)^T Q vt \\ &= r_0^T Q r_0 + (2 r_0^T Q v)t + (v^T Q v)t^2\end{aligned}$$

The quadratic formula can be used to find the real roots, if they exist, and then we can choose the closest intersection which is in front of the camera (i.e., smallest positive real root).

In order to compute the illumination of this point on the quadric, we also need to know the normal of the surface. This will simply be the gradient of this implicit function evaluated at the point of intersection. The gradient in this case is easier to see if we write it in non-homogeneous coordinates. That is, let us write the quadric as

$$x^T A x + B x + C = 0$$

where we can break this down further using the 16 coefficients that we used in the previous section.

$$A = \begin{pmatrix} a & d/2 & f/2 \\ d/2 & b & e/2 \\ f/2 & e/2 & c \end{pmatrix} \quad B = \begin{pmatrix} g & h & i \end{pmatrix} \quad C = (j)$$

In this case, we can see the gradient as $2Ax + B^T$. Note that this is **not** the same as $2Qx$ with x in homogeneous coordinates!