Physical Modeling and Surface Detection

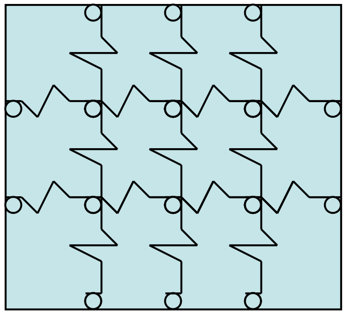
CS116B
Chris Pollett
Mar. 14, 2005.

Outline

- Particle Systems
- Physical Modeling and Visualization
- Classification of Visible Surface Detection Algorithms
- Back Face Detection
- The Depth Buffer Method

Particle Systems

- It is often useful to describe an object as a collection of disjoint pieces.
- Such an object is called a *particle system*.
- Particle systems can be useful for modeling things like smoke, fluids, explosions, grass, etc.
- As an example, to create fireworks, start with a collection of particle spheres drawn at some single fixed point. Shoot the particles out from this point in different, random directions and add gravity.



Physical Modeling

- Non-rigid objects, such as rope, cloth,etc, can be modeled with physically based modeling techniques.
- For instance, a cloth might be modeled as a grid of mass points connected by strings.
- Force equation for such a spring given F = kx. Can model how mass points move when other forces like gravity are applied. Then one can texture map polygons faces of grid of mass points to draw the final object.

Visualization of Data Sets

- Scalar Fields -- function from several dimensions into one. For example, f(x,y) or f(x,y,z).
  - To draw f(x,y) can use elevation grid
  - To draw f(x,y,z) can use pseudo-color methods and assign ranges of values for field different color values.
  - To draw can use contour plots of f(x,y) =c or f(x,y,z)=c for different c's. 2D-case gives isolines; 3d case gives iso-surfaces.

Representing Vector Fields

- Functions which take a vector and return a vector.
- **F**(x,y,z) or **F**(x,y).
  - can draw lines and arrows attached to points (x,y,z), or (x,y).
  - Can use field lines

Representing Tensors

- A tensor of type (p, q) on R^n takes p row vectors of length n and q column vectors of length and outputs a scalar. Alternatively, can think of as taking q column vectors and outputting p row vectors of length n. Map must be linear in each argument.
- Used in talking about materials (stress tensor), fluid dynamics, relativity, etc.
- To draw one can output tensor contractions of the tensor or can use different colored scalar or vector representations superimposed on the same scene.

Classification of Visible Surface Detection Algorithms

- Algorithms for determining which surfaces would be visible on the screen can be broken into two broad categories:
  - Object space methods -- these compare objects or parts of objects to figure out who is in front of who.
  - Image space methods -- these compute visibility point by point at each pixel value on the projection plane.

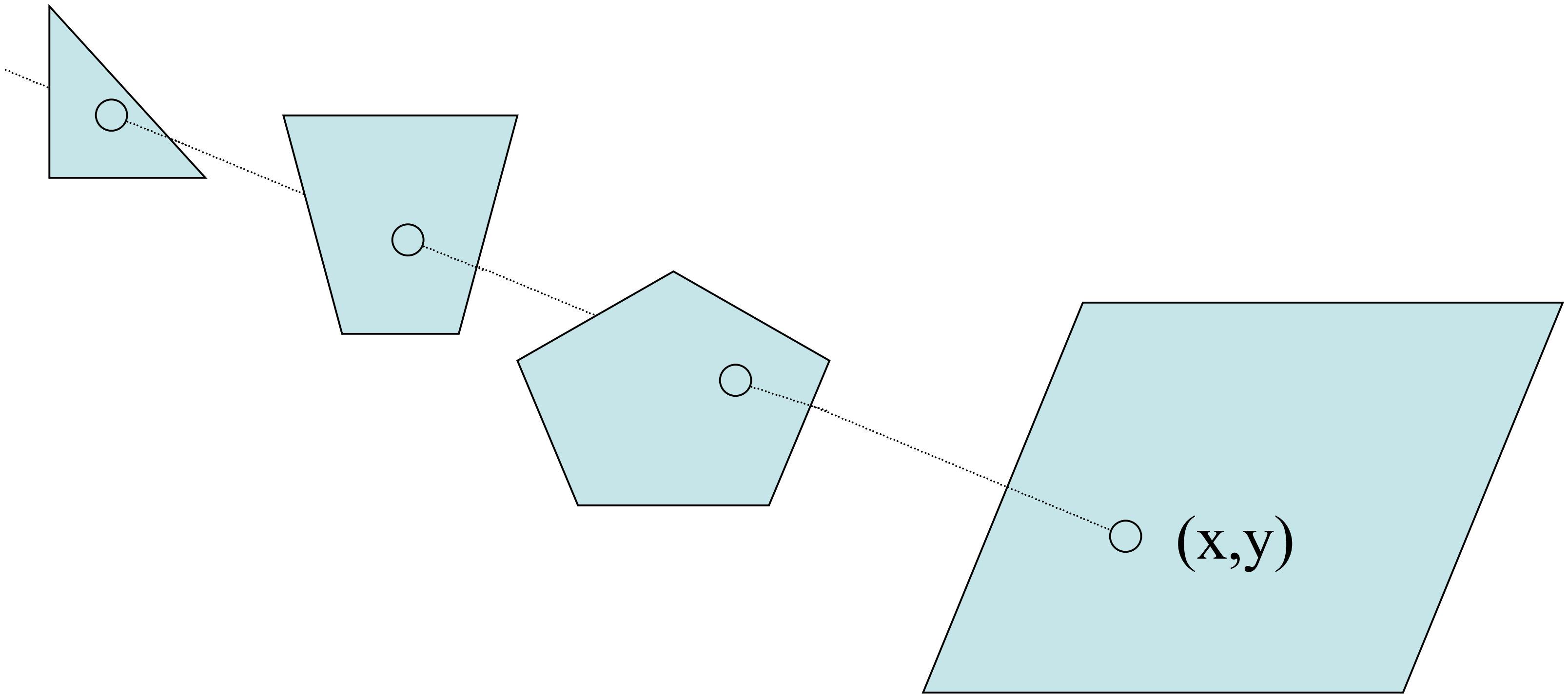
Back Face Detection

- A point (x,y,z) is said to be behind a polygon surface if $Ax + By + Cz + D < 0$ where A,B,C,D determine the plane of the polygon.
- If this point is along the line of sight to the surface, then we must be looking at the back of the polygon.
- Said another way, if **N** is the normal to the polygon and $\mathbf{V}_{\mathbf{view}}$ is the viewing vector from the camera position, then the polygon is a back face if $\mathbf{V}_{\mathbf{view}} \cdot \mathbf{N} > 0$.
- If the object has been converted to projection coordinates, then our viewing direction is parallel to the z axis and only need to consider z component of **N**. So a polygon is a back iff the z component of **N**, C above, is less than 0.

The Depth Buffer Method

- This method compares surface depth values throughout a scene for each pixel position on the projection plane.
- It works for non planar surfaces, but is usually implemented for polygon surfaces.
- Sometimes called *z-buffer method*.

Algorithm



- Let depthBuff(x,y) := 1.0, frameBuff(x,y) = backgndColor; //assume 1.0=far
- Process each polygon in scene one at a time.
  - for each project (x,y) pixel position of a polygon, calculate the depth z.
  - If z<depthBuff(x,y) compute surface color of that polygon, set depthBuff(x,y) =z;
    - frameBuff(x,y) =surfColor(x,y);

More Algorithm

- At (x,y), depth is calculated from the plane equation as: $z= -(Ax+By+D)/C$.
- We want to be able to quickly compute adjacent points on a scan-line. So given z, to calculate depth at (x+1, y) could compute $z' = z - A/C$
- New x' values along an edge of a polygon (changing y value by -1) given by $x' = x - 1/m$, where m is the slope of the line.
- For this x' get: $z' = z + (A/m + B)/C$.
- The above thus describes how to quickly compute along a scan line, then how to move to next line.