Project 2 Clarification

- you don’t have to support relative and absolute camera motion simultaneously
  - OK to reset the view when you switch between modes
  - use ‘m’ to toggle between modes

Reminder: Project Handin

- due 6pm Thursday
- when handing after the deadline, handin has this unfriendly warning message
  - Checking that handin was successful ...
  - /cs/csbox/user FAILED to find user a1b2. Your files DO NOT appear to be handed in successfully
  - Do you want to cancel?
- don’t panic
  - go ahead and complete the handin, do not cancel!
  - your submission will be put in the LATE directory

Review: Bilinear Interpolation

- interpolate quantity along \( L \) and \( R \) edges, as a function of \( y \)
  - then interpolate quantity as a function of \( x \)

Review: Barycentric Coordinates

- weighted combination of vertices
  \[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]
  \[ \alpha + \beta + \gamma = 1 \]
  \[ 0 \leq \alpha, \beta, \gamma \leq 1 \]

Review: Clipping

- analytically calculating the portions of primitives within the viewport
Review: Clipping Lines To Viewport
- combining trivial accepts/rejects
  - trivially accept lines with both endpoints inside all edges of the viewport
  - trivially reject lines with both endpoints outside the same edge of the viewport
  - otherwise, reduce to trivial cases by splitting into two segments

Review: Cohen-Sutherland Line Clipping
- outcodes
  - 4 flags encoding position of a point relative to top, bottom, left, and right boundary

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>y==ymax</th>
<th>y==ymin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OC(p1)== 0 && OC(p2)==0
- trivial accept
(OC(p1) & OC(p2))!= 0
- trivial reject

Review: Polygon Clipping
- not just clipping all boundary lines
- may have to introduce new line segments

Review: Sutherland-Hodgeman Clipping
- for each viewport edge
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped
- for each polygon vertex
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?

Visibility
Reading
- FCG Chapter 7

Rendering Pipeline

Covered So Far
- modeling transformations
- viewing transformations
- projection transformations
- clipping
- scan conversion
- lighting
- shading
- we now know everything about how to draw a polygon on the screen, except visible surface determination

Invisible Primitives
- why might a polygon be invisible?
  - polygon outside the field of view / frustum
    - solved by clipping
  - polygon is backfacing
    - solved by backface culling
  - polygon is occluded by object(s) nearer the viewpoint
    - solved by hidden surface removal
- for efficiency reasons, we want to avoid spending work on polygons outside field of view or backfacing
- for efficiency and correctness reasons, we need to know when polygons are occluded

Back-Face Culling
- most objects in scene are typically “solid”
- rigorously: orientable closed manifolds
  - orientable: must have two distinct sides
  - cannot self-intersect
  - a sphere is orientable since has two sides, ‘inside’ and ‘outside’
  - a Mobius strip or a Klein bottle is not orientable
  - closed: cannot “walk” from one side to the other
  - sphere is closed manifold
  - plane is not
Back-Face Culling
- most objects in scene are typically “solid”
- rigorously: orientable closed manifolds
  - manifold: local neighborhood of all points isomorphic to disc
  - boundary partitions space into interior & exterior

Yes  No
3 3 2

Manifold
- examples of manifold objects:
  - sphere
  - torus
  - well-formed CAD part

Back-Face Culling
- examples of non-manifold objects:
  - a single polygon
  - a terrain or height field
  - polyhedron w/ missing face
  - anything with cracks or holes in boundary
  - one-polygon thick lampshade

Back-Face Culling
- on the surface of a closed manifold, polygons whose normals point away from the camera are always occluded:

note: backface culling alone doesn’t solve the hidden-surface problem!

Back-face Culling: VCS
- first idea: cull if $N_z < 0$
- sometimes misses polygons that should be culled
- better idea: cull if eye is below polygon plane

not rendering backfacing polygons improves performance
- by how much?
  - reduces by about half the number of polygons to be considered for each pixel
Back-face Culling: NDCS

Hidden Surface Removal

Occlusion
- for most interesting scenes, some polygons overlap
- to render the correct image, we need to determine which polygons occlude which

Painter’s Algorithm
- simple: render the polygons from back to front, “painting over” previous polygons
- draw blue, then green, then orange
- will this work in the general case?

Painter’s Algorithm: Problems
- intersecting polygons present a problem
- even non-intersecting polygons can form a cycle with no valid visibility order:

Analytic Visibility Algorithms
- early visibility algorithms computed the set of visible polygon fragments directly, then rendered the fragments to a display:
Analytic Visibility Algorithms

- What is the minimum worst-case cost of computing the fragments for a scene composed of \( n \) polygons?
- Answer: \( O(n^2) \)

Binary Space Partition Trees (1979)

- BSP Tree: partition space with binary tree of planes
- Idea: divide space recursively into half-spaces by choosing splitting planes that separate objects in scene
- Preprocessing: create binary tree of planes
- Runtime: correctly traversing this tree enumerates objects from back to front

Creating BSP Trees: Objects

- For about a decade (late 60s to late 70s) there was intense interest in finding efficient algorithms for hidden surface removal
- We'll talk about two:
  - Binary Space-Partition (BSP) Trees
  - This time
  - Warnock's Algorithm
  - Next time
Creating BSP Trees: Objects

- no bunnies were harmed in previous example
- but what if a splitting plane passes through an object?
- split the object; give half to each node

Splitting Objects

- tree creation independent of viewpoint
- preprocessing step
- tree traversal uses viewpoint
- runtime, happens for many different viewpoints
- each plane divides world into near and far
- for given viewpoint, decide which side is near and which is far
- check which side of plane viewpoint is on
- recursive algorithm
  - recurse on far side
  - draw object
  - recurse on near side

Traversing BSP Trees

- query: given a viewpoint, produce an ordered list of (possibly split) objects from back to front:

```
renderBSP(BSPtree *T)
BSPtree *near, *far;
if (eye on left side of T->plane)
  near = T->left; far = T->right;
else
  near = T->right; far = T->left;
renderBSP(far);
if (T is a leaf node)
  renderObject(T)
renderBSP(near);
```
BSP Trees: Viewpoint B

BSP Tree Traversal: Polygons
- split along the plane defined by any polygon from scene
- classify all polygons into positive or negative half-space of the plane
  - if a polygon intersects plane, split polygon into two and classify them both
  - recurse down the negative half-space
  - recurse down the positive half-space

BSP Demo
- useful demo: [http://symbolcraft.com/graphics/bsp](http://symbolcraft.com/graphics/bsp)

Summary: BSP Trees
- pros:
  - simple, elegant scheme
  - correct version of painter’s algorithm back-to-front rendering approach
  - was very popular for video games (but getting less so)
- cons:
  - slow to construct tree: $O(n \log n)$ to split, sort
  - splitting increases polygon count: $O(n^2)$ worst-case
  - computationally intense preprocessing stage restricts algorithm to static scenes