Lecture n+1:
Recursive Ray Tracer2:
Advanced Techniques and Data Structures
Review

- Ray Intersect (Assignment 4): questions / comments?

- Review of Recursive Ray Tracer:
  - Two (or three) types of secondary rays

- Lighting equation

- Blending and texture mapping

- The value of Epsilon...
Questions?
Glowing Ball

- Now that you see how complicated (and slow) real rendering is, how do you do some common techniques that appear “cool” but run in real time?
  - For example, a glowing floating ball?
  - [https://www.youtube.com/watch?v=kAG1MbDeDps](https://www.youtube.com/watch?v=kAG1MbDeDps)

- Short answer: fake it with texture maps
Advanced Techniques (Shadows 1/2)

- Total hack: Project object on ground plane and render it as black polygon. Eye less sensitive to exact geometry than to contrast of shadow.
- Projective shadows (shadow mapping)
  - Use orthographic or perspective matrix as CTM to project object (shadow caster) against another object; render the shadow object as another primitive.
  - Good for single light scenes with shadows cast on planes.
- Shadow volumes
  - Extrude silhouette volume from object (shadow caster) to infinity in the direction opposite the light source.
  - Silhouette edge is made up of adjacent polygons, any point within the volume is in shadow.

Implemented by former cs123 ta Kevin Egan and former PhD student and book co-author Prof. Morgan McGuire, on nVidia chip.
Advanced Techniques (Shadows 2/2)

- **Shadow Maps**
  - **Transform camera to each directional light source**
    - Render the scene from the light source POV, keeping the same far clipping plane, update z-buffer only
    - Read the z-buffer back and apply as a texture (shadow map) to the scene projected from the light’s POV
    - Render the scene from the eye point. At each pixel, if distance from light is greater than in the shadow map, the object is in shadow.
    - Major aliasing problem, but implemented in hardware

By keeping the same far clipping plane, relative distances in Z are preserved.
Advanced Techniques (Environment Mapping)

- A way to render reflections (also called reflection mapping)
  - Simulates reflections with a texture map applied to the enclosing sphere or cube of an object
  - Texture map can either be an existing image or a rendering of the scene from the object’s perspective

- Cube environment map is created by stitching together 6 textures, each generated by rendering the scene from the object’s POV
  - Ray from eye is reflected over surface normal and intersected with the environment map to get texture coordinates.

Shiny metal effect using environment map.
Overview: There are several hacks to approximate the lighting and shading of complex geometry without actually describing the geometry – only used for fine detail

- Perturb geometric normals with bump map
- Approximate geometry at render time with displacement map
- Replace normals of simpler geometry with normal map
Surface Detail (Bump Mapping)

- Texture mapping a rough surface onto an object doesn’t achieve desired effect. Illumination is too uniform.
- Blinn’s technique: Use an array of values (stored in a texture called the “bump map”) to perturb the surface normals by calculating gradient of bump map and adding it to the surface normal.
- Evaluate lighting model with perturbed normals.
- Effect is convincing, but objects still look smooth at silhouette edges.
Bump Mapping Example

- The tin foil on this Hershey’s Kiss looks too smooth
Bump Mapping Example

- Bump Mapped: Tin foil look with no increase in model complexity.
- Edges still don’t look quite right
Normal Mapping

- Similar to bump mapping, but better for highly detailed surfaces with drastically varying normals
  - Bump mapping uses a single channel height field to perturb normals, while normal mapping uses a three channel map to actually replace existing surface normal
  - $x$, $y$, and $z$ components of the desired normal, stored in the $r$, $g$, and $b$ channels of an image
- Level of detail is now limited by normal map resolution, not number of triangles
  - Runtime surface normal determination is as easy as a texture lookup
- Limitations:
  - Same as bump map, silhouette edges lack the added detail
  - Though easy to use at runtime, normal maps are difficult to generate
Normal Mapping Examples

Render showing actual, simple underlying geometry

When a normal map is applied, there appears to be much more detail in the model than actually is

- Normal mapping can completely alter the perceived geometry of a model
Normal Map Creation (1/3)

- While bump maps can be simply drawn, normal maps hold meaningful data in three overlapping channels.

Normal map for a face. Red, green, and blue channels correspond to $x$, $y$, and $z$ normal components.

- Normal maps are not visually intuitive like height maps. How are they created?
Normal Map Creation (2/3)

- Normal maps must be generated by specialized software, such as Pixologic’s Zbrush (www.pixologic.com)
  - A high resolution model is created by an artist and its normals are used to generate a normal map for a low resolution model.
Normal Mapping Creation (3/3)

How it works:
- Artist creates a high resolution model with complex geometry (video)
- High resolution model simplified using mesh decimation to a model with much fewer triangles

Normal map generation:
- Let $M$ be the complex original mesh and $S$ the simplified mesh
- To generate the normal to map to a point $P_S$ on $S$, find the point $P_M$ on $M$ such that the Euclidean distance between $P_S$ and $P_M$ is minimal.
- $P_M$ can be found by having each face of $M$ stored in a grid and indexing into this grid with coordinates collected during mesh simplification. We find the minimum perpendicular distance from $P_S$ to each of $M$’s nearby facets.
- Compute the normal at $P_M$ by averaging normals at surrounding vertices. Place this value in triangular normal map for the single face.
- Finally, stitch these triangular normal maps together to create a normal map for the entire simple surface.
- Note: The resolution at which normals are sampled is user defined (again, time vs quality tradeoff)

See SIGGRAPH paper: “A general method for preserving attribute values on simplified meshes” by Cignoni, Montani, Rocchini, Scopigno
Other Surface Detail Techniques (1/3)

- **Displacement Map:**
  - The actual geometric position of points over the surface are *displaced* along the surface normal according to the values in a texture.


The displacement map used to render the scene. The intensity value indicates the height at a given point.
Other Surface Detail Techniques (2/3)

- **Parallax Mapping:**
  - Displaces texture coordinates of a surface point by a function of the view angle relative to the surface normal and the value on the height map at that point.

- **Steep Parallax Mapping:**
  - Similar scheme to produce self-occlusion self-shadows of parallax mapped surface; More efficient and reduces swim effect for high frequency maps (Morgan McGuire Ph.D. ’06 and Max McGuire)
Other Surface Detail Techniques (3/3)

Texture Mapped

Normal Mapped: Increased depth in rock wall

Parallax Mapped: Increased depth of whole texture but unwanted “swim” effect. Textures look like they should be 3D but they stay embedded in surface, unsettling to the eye when moving

Steep Parallax Mapped: Very realistic depth in whole picture. Lion head, columns cast self shadows

Questions?
How to Speed Up My Ray Tracer?!
What Would You Do?

- In the example below, we have to compute 8 intersection tests to find that we intersected with nothing

- Is there a way to speed this up?
Screen-Based Approach

- We can project the objects onto the screen, and compute the bounding boxes in 2D.
  - No need to cast rays where the screen is empty

- If we use OpenGL to render the scene once, we can (roughly) determine what object(s) we need to intersect with!
What Would You Do?

- In the example below, we have to compute 8 intersection tests to find the nearest object.
- Is there a way to speed this up?
Bounding Volumes

- Use bounding volumes (that are easier/cheaper to compute intersection tests for)
  - Spheres and boxes are most common
  - It’s typical to use axis-aligned bounding boxes (makes math easier – no coordinate transform)
  - Works very well for tight bounding boxes

- Do this recursively to get a set of hierarchical bounding boxes (bottom-up or top-down)
  - Can be applied to the scene graph if the objects in the scene are organized spatially
Creating a Bounding Volume Hierarchy (BVH)

- Using a bottom-up approach
Bounding Volume Hierarchy (BVH) Example
Different Ways To Partition a Space

- Bounding volume hierarchy
- Grid
- Octree
- kd-Tree (k-dimensional tree)
- BSP-Tree (binary space partitioning trees)
Grids

- Divide the world into uniform cells
  - Each cell stores the objects contained within it
- Use line-scanning algorithm to traverse
- Intersect with each cell along the way
Grids

- **Problems:**
  - Choosing the right number of cells isn’t the most obvious
    - Too many cells -> need to check a lot of each ray
    - Too few cells -> speed up can be reduced (too many objects in each one)
  - Some cells could be empty, while others have tons of objects within it

- Note that this approach is not hierarchical, which means that we don’t get the nice $O(\log n)$ benefit

- Need to be careful in deciding what objects belong to a cell (vertex, edge, face)

- But very useful for many reasons:
  - Easy to implement in hardware
  - Good for an animated scene (an object moving to a different cell doesn’t affect the other objects or other cells)
Octree

- Basically this is an adaptive grids approach
  - Generate more cells when needed, but leave the white spaces alone
  - Cells are still axis-aligned
- Cells will no longer have the same size

2-D example (called a quadtree)
Octree

-Constructed using a top-down approach

-Recursively split the world into 8 cells
  - Terminates when the depth is too high, or
  - Terminates when the number of objects per cell falls below a threshold

-Similar problem as grids, need to be careful in determining which objects belong to which cells
Octree

- Start at the root node (cell), if it intersects, recursively check all 8 children (or 4 for quadtrees).
- If a leaf node (cell) is empty, terminate. Otherwise, check all objects within the cell.
Octree

- For each ray, the expected runtime is $O(\log n)$
- Problems:
  - Updating an octree could be tricky (for an animated scene)
  - Finding neighbors of a cell isn’t trivial
  - For scenes that have objects distributed non-uniformly, the depth could become very high
**Kd-Tree**

- Improves upon octrees in that it is a little more “intelligent” in finding the split points
  - Skips large swaths of empty spaces early
  - Zeros in on areas with high complexity (lots of objects)

- In the worst case, a kd-Tree will be exactly like a quadtree (for \( k=2 \)), or an octree (for \( k=3 \))

- Like grids and octrees, kd-trees also use axis-aligned bounding boxes (for checking intersection is fast)

- Unlike octrees, kd-Trees split in one dimension at a time
kd-Tree: Choosing a Split Plane

- (Recursively) splitting a middle results in a quadtree
- During traversal, cost of entering the right side is much higher
kd-Tree: Choosing a Split Plane

- Splitting at the medium balances out the cost, but
- During traversal, the chance of entering the left hand side is much higher (the area is larger)
kd-Tree: Choosing a Split Plane

- Cost-optimized split: attempts to balance the cost of entering and the probability of entering the two sides.
- Isolates the complex geometry quickly, creating large, (mostly) empty spaces that can be efficiently computed and discarded.
Further subdivision does not result in big gains (compared to octrees)

But the point is that we isolated the complex geometry in one split (which would have taken octree much longer to find)
kd-Tree: Caution with Edge Cases

- Notice the ray from left to right
  - When it enters F, it will do an intersection test with the green rectangle
  - You will be tempted to say that you’re done and return the green rectangle as the nearest object (because F is in front of G)
  - However, one can see that the clear nearest object is the yellow rectangle...

- Solution, check the t-value against the bounding box t-value
  - Which will show that the t-value for the green rectangle is behind F.
BVH

Octree

Cells

kd-Tree
Binary Space Partition (BSP) Tree

- Not really used for ray tracer, but for visible surface determination
  - This algorithm replaces the use of z-buffers
  - Fast to construct. Used in video games (like Quake 3)
  - Similar to kd-Tree, but NOT axis-aligned

BSP-0: Initial Scene
BSP Trees

BSP-1: Choose any polygon (e.g., polygon 3) and subdivide others by its plane, splitting polygons when necessary.
BSP Trees

BSP-2: Process front sub-tree recursively
BSP Trees

BSP-3: Process back sub-tree recursively
BSP Trees

BSP-4: An alternative BSP tree with polygon 5 at the root
BSP-Trees

- Remember that this is used for determining the order in which objects need to be drawn on screen.

- So the order in which the objects are drawn will be from back to front, but takes into account of the direction of the face and where the eye is (recursively).
BSP-Trees

```c
void BSP_displayTree(BSP_tree* tree) {
    location = tree->find_location(eye);
    if ( location > 0 ) // eye in front of object
        if ( viewer is in front of root ) {
            BSP_displayTree(tree->backChild);
            displayPolygon(tree->root);
            BSP_displayTree(tree->frontChild);
        }
    else { // eye behind the object
        BSP_displayTree(tree->frontChild);
        // Ignore next line if back-face
        // culling desired
        displayPolygon(tree->root);
        BSP_displayTree(tree->backChild);
    }
}
```
Summary

- Which data structure would be best to use in the following situations?
  - An animation where only the camera moves?
    - kd-Tree
  - A fast-paced video game with lots of interactive, moving objects?
    - regular grid or BSP-Tree
  - A very balanced and even scene with objects uniformly distributed throughout the whole scene?
    - octree
  - Rendering scenes in a Pixar-like movie?
    - depends on the shot; most likely a combination of techniques are used
  - Your ray-tracing assignment if you are really fed up with your rendering speed?
    - kd-Tree is best, but balance your decision with difficulty in implementing the solution...