Correction: P1 Hall of Fame: Winner
Sung-Hoon (Nick) Kim

Further Clarification: Blinn-Phong Model
• only change vs Phong model is to have the specular calculation to use \((h \cdot n)\) instead of \((v \cdot f)\)
• full Blinn-Phong lighting model equation has ambient, diffuse, specular terms
  \[
  I_{\text{total}} = k_i I_{\text{ambient}} + \sum_{m=1}^{N} I_{\text{spec}}(k_d (n \cdot l_i) + k_s (n \cdot h_i)^m) 
  \]
• just like full Phong model equation
  \[
  I_{\text{total}} = k_i I_{\text{ambient}} + \sum_{m=1}^{N} I_{\text{spec}}(k_d (n \cdot l_i) + k_s (n \cdot h_i)^m) 
  \]

Review: Sutherland-Hodgeman Clipping
• for each viewport edge
  • clip the polygon against the edge equation for new vertex list
  • after doing all edges, the polygon is fully clipped

Review: Polygon Clipping
• not just clipping all boundary lines
• may have to introduce new line segments
• for each polygon vertex
  • decide what to do based on 4 possibilities
  • is vertex inside or outside?
  • is previous vertex inside or outside?

Review: Painter’s Algorithm
• 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
• 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
Splitting 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

Traversing BSP Trees
• 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 independently for each tree vertex
  • tree traversal differs depending on viewpoint!
• recursive algorithm
  • recurse on far side
  • draw object
  • recurse on near side

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

Interpolating Z
- barycentric coordinates
- interpolate Z like other planar parameters

Z-Buffer
- store (r, g, b, z) for each pixel
- typically 8+8+8+24 bits, can be more

Depth Test Precision
- reminder: perspective transformation maps eye-space (view) z to NDC z

Correction: Ortho Camera Projection
- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away z values
- x and y coordinates do not change with respect to z in this projection

Clarification: BSP Demo
- order of insertion can affect half-plane extent

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

The Z-Buffer Algorithm (mid-70’s)
- BSP trees proposed when memory was expensive
  - first 512x512 framebuffer was >$50,000!
- Ed Catmull proposed a radical new approach called z-buffering
- the big idea:
  - resolve visibility independently at each pixel

The Z-Buffer Algorithm
- we know how to rasterize polygons into an image discretized into pixels:

The Z-Buffer Algorithm
- what happens if multiple primitives occupy the same pixel on the screen?
  - which is allowed to paint the pixel?

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

The Z-Buffer Algorithm
- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to \( \infty \)
  - when rasterizing, interpolate depth (Z) across polygon
  - check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
  - don’t write pixel if its Z value is more distant than the Z value already stored there

Z-Buffer Demo
- useful demo:
  - http://symbolcraft.com/graphics/bsp

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

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

Depth Test Precision
- reminder: perspective transformation maps eye-space (view) z to NDC z

Correction: Ortho Camera Projection
- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away z values
- x and y coordinates do not change with respect to z in this projection

The Z-Buffer Algorithm
- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to \( \infty \)
  - when rasterizing, interpolate depth (Z) across polygon
  - check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
  - don’t write pixel if its Z value is more distant than the Z value already stored there
Depth Test Precision
- therefore, depth-buffer essentially stores $1/z$, rather than $z$
- issue with integer depth buffers
  - high precision for near objects
  - low precision for far objects

Z-Buffer Cons
- poor for scenes with high depth complexity
  - need to render all polygons, even if most are invisible
  - shared edges are handled inconsistently
    - ordering dependent

Z-Buffer Pros
- simple!!!
- easy to implement in hardware
  - hardware support in all graphics cards today
- polygons can be processed in arbitrary order
- easily handles polygon interpenetration
- enables deferred shading
  - rasterize shading parameters (e.g., surface normal) and only shade final visible fragments

Depth Test Precision
- low precision can lead to depth fighting for far objects
  - two different depths in eye space get mapped to same depth in framebuffer
  - which object "wins" depends on drawing order and scan-conversion
- gets worse for larger ratios $f/r$
- rule of thumb: $f > 1000$ for 24 bit depth buffer
- with 16 bits cannot discern millimeter differences in objects at 1 km distance
- demo: sjbaker.org/steve/omniv/love_your_z_buffer.html

Z-Buffer Algorithm Questions
- how much memory does the Z-buffer use?
  - (e.g. 1280x1024x32 bits)
  - requires fast memory
    - Read-Modify-Write in inner loop
  - hard to simulate translucent polygons
    - we throw away color of polygons behind closest one
    - works if polygons ordered back-to-front
  - extra work throws away much of the speed advantage
  - rule of thumb: $10^{-3}$
- does the image rendered depend on the drawing order?
  - early in pipeline
  - after clipping
- how does Z-buffer load scale with visible polygons? with framebuffer resolution?
  - performs late in rendering pipeline
- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
  - performed late in rendering pipeline

Hidden Surface Removal
- two kinds of visibility algorithms
  - object space methods
  - image space methods

Object Space Algorithms
- determine visibility on object or polygon level
  - using camera coordinates
  - resolution independent
    - explicitly compute visible portions of polygons
  - early in pipeline
  - after clipping
  - requires depth-sorting
  - painter's algorithm
  - BSP trees

Projective Rendering Pipeline
- object world viewing clipping
  - OCS - object coordinate system
  - WCS - world coordinate system
  - VCS - viewing coordinate system
  - CCS - clipping coordinate system
  - DCS - device coordinate system

Image Space Algorithms
- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
  - performed late in rendering pipeline

Rendering Pipeline
- object world viewing clipping
  - OCS - object coordinate system
  - WCS - world coordinate system
  - VCS - viewing coordinate system
  - CCS - clipping coordinate system
  - DCS - device coordinate system

Read-Modify-Write in inner loop
- extra work throws away much of the speed advantage

Model/View Transformation
- object world viewing
- OCS - object coordinate system
  - glTranslatef(x,y,z)
  - gluLookAt(x,y,z)
- WCS - world coordinate system
  - gluPerspective(f/ w, f/r, z/ w, z/r)
- VCS - viewing coordinate system
  - gluViewport(x,y,a,b)
- CCS - clipping coordinate system
  - glFrustum(x,y,a,b)
- DCS - device coordinate system
  - glVertex3f(x,y,z)

Depth Test Precision
- low precision can lead to depth fighting for far objects
  - two different depths in eye space get mapped to same depth in framebuffer
  - which object "wins" depends on drawing order and scan-conversion
- gets worse for larger ratios $f/r$
- rule of thumb: $f > 1000$ for 24 bit depth buffer
- with 16 bits cannot discern millimeter differences in objects at 1 km distance
- demo: sjbaker.org/steve/omniv/love_your_z_buffer.html