Midterm 2 Review

Week 10, Fri Mar 18

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
News

- homework 3 handed back, solutions out
- grades posted for P2, H2, H3
Common Homework Mistakes

- homework 2
  - multiplying colors: do it component-wise, not dot product or cross product
  - remember that chromaticity “horseshoe” diagram shows only hue/saturation, not intensity
  - Bresenham algorithm: must define dx, dy

- homework 3
  - line clipping: remember to discard wrong segment after each intersection
  - poly clipping: make sure to explicitly traverse vertex list
Common Homework Mistakes

- homework 3
  - BSP trees:
    - construction: line segments are the objects you’re drawing
    - construction: should divide only subspace with each new plane, not all of space
    - traversal: must decide separately at each tree node whether to take left or right path based on eye position
  - pipeline: Gouraud shading at both lighting and scan conversion
Review: Collision Detection Algorithms

- naive very expensive: $O(n^2)$
- primary factor: geometry of colliding objects
- secondary factor: way in which objects move
- other factors: speed, simplicity, robustness
- optimizations
  - if more than one test available, with different costs: how do you combine them?
  - how do you avoid unnecessary tests?
  - how do you make tests cheaper?
Review: Fundamental Design Principles

- *fast simple tests first*, eliminate many potential collisions
- exploit *locality*, eliminate many potential collisions
- use as much *information* as possible about geometry
- exploit *coherence* between successive tests
- specific techniques
  - collision proxies
  - spatial data structures to localize finding precise collision times
Review: Collision Proxies

- **collision proxy** *(bounding volume)* is piece of geometry used to represent complex object for purposes of finding collision

- good proxy: cheap to compute collisions for, tight fit to the real geometry

- proxies exploit facts about human perception
  - we are extraordinarily bad at determining correctness of collision between two complex objects
  - the more stuff is happening, and the faster it happens, the more problems we have
Review: Trade-off in Choosing Proxies

increasing complexity & tightness of fit

decreasing cost of (overlap tests + proxy update)
Review: Spatial Data Structures

- uniform grids
- bounding volume hierarchies
- octrees
- BSP trees
- kd-trees
- k-dops
Review: Exploiting Coherence

- player normally doesn’t move far between frames
- track incremental changes, using previous results instead of doing full search each time
- keep track of entry and exit into cells through portals
  - probably the same cells they intersect now
  - or at least they are close
Review: Precise Collisions

- hacked clean up
  - simply move position so that objects just touch, leave time the same

- interval halving
  - binary search through time to find exact collision point and time
Review: Temporal Sampling

- temporal sampling
  - aliasing: can miss collision completely!
Review: Managing Fast Moving Objects

- several ways to do it, with increasing costs
- movement line: test line segment representing motion of object center
  - pros: works for large obstacles, cheap
  - cons: may still miss collisions. how?
- conservative prediction: only move objects as far as you can be sure to catch collision
  - increase temporal sampling rate
  - pros: will find all collisions
  - cons: may be expensive, how to pick step size
- space-time bounds: bound the object in space and time, check bound
  - pros: will find all collisions
  - cons: expensive, must bound motion
Prediction and Bounds

- **conservative motion**
  - assume maximum velocity, smallest feature size
  - largest conservative step is smallest distance divided by the highest speed - clearly could be very small
  - other more complex metrics are possible

- **bounding motion**
  - assume linear motion
  - find radius of bounding sphere
  - build box that will contain that sphere for frame step
  - also works for ballistic and some other predictable motions

- simple alternative: just miss the hard cases
  - player may not notice!
Collision Response

- for player motions, often best thing to do is move player tangentially to obstacle
- do recursively to ensure all collisions caught
  - find time and place of collision
  - adjust velocity of player
  - repeat with new velocity, start time, start position (reduced time interval)
- handling multiple contacts at same time
  - find a direction that is tangential to all contacts
Related Reading

- Real-Time Rendering
  - Tomas Moller and Eric Haines
  - on reserve in CICSR reading room
Midterm 2 Review
Logistics

- policies
  - leave backpacks etc at front of room
  - must have student photo ID face up on desk
    - cannot take exam without photo ID
  - one piece of 8.5”x11” paper allowed
    - one side handwritten
  - no other books or notes
  - nonprogrammable calculator OK
Topics Possibly Covered

- color
- rasterization/scan conversion
- clipping
- visibility / hidden surfaces
- texturing
- procedural approaches
- advanced rendering
- sampling/antialiasing
- animation
- picking
- rendering pipeline
Color
Review: Simple Model of Color

- based on RGB triples
- component-wise multiplication of colors
  - \((a_0, a_1, a_2) \times (b_0, b_1, b_2) = (a_0 \times b_0, a_1 \times b_1, a_2 \times b_2)\)
Review: Trichromacy and Metamers

- three types of cones
- color is combination of cone stimuli
  - metamer: identically perceived color caused by very different spectra
Review: Color Constancy

Do they match?

Image courtesy of John McCann
Review: Measured vs. CIE Color Spaces

- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes

- transformed basis
  - “imaginary” lights
  - all positive, unit area
  - Y is luminance
Review: Device Color Gamuts

- compare gamuts on CIE chromaticity diagram
- gamut mapping
Review: RGB Color Space

- define colors with \((r, g, b)\) amounts of red, green, and blue
  - used by OpenGL

- RGB color cube sits within CIE color space
  - subset of perceivable colors
Review: HSV Color Space

- **hue**: dominant wavelength, “color”
- **saturation**: how far from grey
- **value/brightness**: how far from black/white
Review: YIQ Color Space

- YIQ is the color model used for color TV in America. Y is brightness, I & Q are color
  - same Y as CIE, backwards compatibility with black and white TV
  - blue is more compressed

\[
\begin{bmatrix}
Y \\ I \\ Q
\end{bmatrix} =
\begin{bmatrix}
0.30 & 0.59 & 0.11 \\
0.60 & -0.28 & -0.32 \\
0.21 & -0.52 & 0.31
\end{bmatrix}
\begin{bmatrix}
R \\ G \\ B
\end{bmatrix}
\]
Review: Gamma Correction

\[ \gamma_{DS} = \gamma_{D} \left( \frac{1}{\gamma_{OS}} \right) \]
Scan Conversion
Review: Midpoint Algorithm

- moving incrementally along x direction
  - draw at current y value, or move up to y+1?
    - check if midpoint between two possible pixel centers above or below line

- candidates
  - top pixel: \((x+1, y+1)\),
  - bottom pixel: \((x+1, y)\)

- midpoint: \((x+1, y+.5)\)
- check if midpoint above or below line
  - below: top pixel
  - above: bottom pixel

- assume \(x_1 < x_2\), slope \(0 < \frac{dy}{dx} < 1\)
Review: Bresenham Algorithm

y=y0; e=0;
for (x=x0; x <= x1; x++) {
    draw(x,y);
    if (2(e+dy) < dx) {
        e = e+dy;
    } else {
        y=y+1;
        e=e+dy-dx;
    }
}

- all integer arithmetic
- cumulative error function

y=y0; eps=0
for ( int x = x0; x <= x1; x++ ){
    draw(x,y);
    eps += dy;
    if ( (eps << 1) >= dx ){
        y++;  eps -= dx;
    }
}
Review: Flood Fill

- draw polygon edges, seed point, recursively set all neighbors until boundary is hit to fill interior

- drawbacks: visit pixels up to 4x, per-pixel memory storage needed
Review: Scanline Algorithms

- set pixels inside polygon boundary along horizontal lines one pixel apart
- use bounding box to speed up
Review: Edge Walking

- basic idea:
  - draw edges vertically
    - interpolate colors down edges
  - fill in horizontal spans for each scanline
    - at each scanline, interpolate edge colors across span
idea: use a parity test

for each scanline
  edgeCnt = 0;
  for each pixel on scanline (l to r)
    if (oldpixel->newpixel crosses edge)
      edgeCnt ++;
    // draw the pixel if edgeCnt odd
    if (edgeCnt % 2)
      setPixel(pixel);
Interpolation
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

\[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]
\[ \alpha + \beta + \gamma = 1 \]
\[ 0 \leq \alpha, \beta, \gamma \leq 1 \]

\[ a_1 = \frac{c_1}{c_1 + c_2} \frac{b_1}{b_1 + b_2} \]
\[ a_2 = \frac{c_2}{c_1 + c_2} \frac{d_2}{d_1 + d_2} + \frac{c_1}{c_1 + c_2} \frac{b_2}{b_1 + b_2} \]
\[ a_3 = \frac{c_2}{c_1 + c_2} \frac{d_1}{d_1 + d_2} \]
Clipping
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

- $OC(p_1) == 0$ && $OC(p_2) == 0$
  - trivial accept

- $(OC(p_1) \ & \ OC(p_2)) != 0$
  - 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 in edge list
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?
Review: Sutherland-Hodgeman Clipping

- edge from $p[i-1]$ to $p[i]$ has four cases
  - decide what to add to output vertex list

```
inside  outside  inside  outside  inside  outside  inside  outside
```

- $p[i]$ output
- $i$ output
- no output
- $i$ output $p[i]$ output
Visibility
Review: Invisible Primitives

**why might a polygon be invisible?**

- polygon outside the field of view / frustum
  - solved by clipping
- polygon is back-facing
  - solved by backface culling
- polygon is occluded by object(s) nearer the viewpoint
  - solved by hidden surface removal
Review: Back-Face Culling

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

Note: backface culling alone doesn’t solve the hidden-surface problem!
Review: Back-face Culling

VCS

\[
\text{culling } N_Z < 0
\]

sometimes

misses polygons that

should be culled

instead, cull if eye is

below polygon plane

NDCS

\[
\text{works to cull if } N_Z > 0
\]
Review: Painter’s Algorithm

- draw objects from back to front
- problems: no valid visibility order for
  - intersecting polygons
  - cycles of non-intersecting polygons possible
Review: BSP Trees

- preprocess: create binary tree
  - recursive spatial partition
  - viewpoint independent
Review: BSP Trees

- **runtime**: correctly traversing this tree enumerates objects from back to front
  - viewpoint dependent
    - check which side of plane viewpoint is on
    - draw far, draw object in question, draw near
- **pros**
  - simple, elegant scheme
  - works at object or polygon level
- **cons**
  - computationally intense preprocessing stage
  - restricts algorithm to static scenes
Review: Warnock’s Algorithm

- start with root viewport and list of all objects
- recursion:
  - clip objects to viewport
  - if only 0 or 1 objects
    - done
  - else
    - subdivide to new smaller viewports
    - distribute objects to new viewpoints
    - recurse
Review: Warnock’s Algorithm

- termination
  - viewport is single pixel
  - explicitly check for object occlusion
- single-pixel case common in high depth complexity scenes
Review: 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
Review: Object vs. Image Space

- object space
  - determine visibility on object or polygon level
  - resolution independent, VCS / NDC coords
  - early in pipeline
  - requires depth sorting objects/polygons

- image space
  - determine visibility at viewport or pixel level
  - resolution dependent, screen coords
  - late in pipeline
Textures
Review: Surface Texture

- define texture pattern over (s,t) domain
  - image – 2D array of “texels”
- assign (s,t) coordinates to each point on object surface
Review: Example Texture Map

```
glTexCoord2d(0,0);
glVertex3d (-x, -y, -z);
glTexCoord2d(1,1);
glVertex3d (-x, y, z);
```

Texture  Object  Mapped Texture

```
glTexCoord2d(0,0);
glVertex3d (-x, -y, -z);
```
Review: Texture

- action when $s$ or $t$ is outside $[0…1]$ interval
  - tiling
  - clamping

- texture matrix stack
  
  ```c
  glMatrixMode( GL_TEXTURE );
  ```
Review: Example Texture Map

glTexCoord2d(4, 4);
glVertex3d (x, y, z);

(4,4)
|  +  |
|---|---|
(0,0)  (4,0)

Texture Object

(0,0)  (0,4)

Mapped Texture

----

glTexCoord2d(1, 1);
glVertex3d (x, y, z);

(1,0)
|  +  |
|---|---|
(0,0)  (1,1)

Texture Object

(0,0)  (0,1)

Mapped Texture

-----
Review: Perspective Correct Interpolation

- screen space interpolation incorrect

\[ s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2} \]
Review: Reconstruction

- how to deal with:
  - **pixels** that are much larger than **texels**?
    - apply filtering, “averaging”
  - **pixels** that are much smaller than **texels**?
    - interpolate
Review: MIPmapping

- image pyramid, precompute averaged versions

Without MIP-mapping

With MIP-mapping
Review: Bump Mapping: Normals As Texture

- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal
Review: Displacement Mapping

- bump mapping gets silhouettes wrong
  - shadows wrong too
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface
Review: Environment Mapping

- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture
Review: Sphere Mapping

- texture is distorted fish-eye view
  - point camera at mirrored sphere
Review: Cube Mapping

- 6 planar textures, sides of cube
  - point camera outwards to 6 faces
Review: Volumetric Texture

- define texture pattern over 3D domain - 3D space containing the object
  - texture function can be digitized or procedural
  - for each point on object compute texture from point location in space
- 3D function $\rho(x,y,z)$
Procedural Approaches
Procedural Textures

- generate “image” on the fly, instead of loading from disk
  - often saves space
  - allows arbitrary level of detail
function marble(point)
    x = point.x + turbulence(point);
    return marble_color(sin(x))
Review: Perlin Noise

- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes
Review: Generating Coherent Noise

- just three main ideas
  - nice interpolation
  - use vector offsets to make grid irregular
  - optimization
    - sneaky use of 1D arrays instead of 2D/3D one
Review: Particle Systems

- changeable/fluid stuff
  - fire, steam, smoke, water, grass, hair, dust, waterfalls, fireworks, explosions, flocks

- life cycle
  - generation, dynamics, death

- rendering tricks
  - avoid hidden surface computations
Review: Other Procedural Approaches

- fractal landscapes
- L-systems
Advanced Rendering
Review: Simple Ray Tracing

- view dependent method
  - cast a ray from viewer’s eye through each pixel
  - compute intersection of ray with first object in scene
  - cast ray from intersection point on object to light sources
Review: Recursive Ray Tracing

- ray tracing can handle
  - reflection (chrome)
  - refraction (glass)
  - shadows
- spawn secondary rays
  - reflection, refraction
    - if another object is hit, recurse to find its color
  - shadow
    - cast ray from intersection point to light source, check if intersects another object
Review: Subsurface Scattering

- light enters surface, bounces around inside, leaves at *different* location on the surface
Review: Radiosity

- conserve light energy in a volume
  - model light transport until convergence
  - solution captures diffuse-diffuse bouncing of light
- view independent technique
  - calculate solution for entire scene offline
  - browse from any viewpoint in realtime
Review: Radiosity

- divide surfaces into small patches
- loop: check for light exchange between all pairs
  - form factor: orientation of one patch wrt other patch (n x n matrix)
Review: Image-Based Rendering

- store and access only pixels
  - no geometry, no light simulation, ...
- input: set of images
- output: image from new viewpoint
  - surprisingly large set of possible new viewpoints
Sampling/Antialiasing
Review: Image As Signal

- 1D slice of raster image
  - discrete sampling of 1D spatial signal
- theorem
  - any signal can be represented as an (infinite) sum of sine waves at different frequencies

![Diagram of intensity vs. pixel position across scanline](image)
Review: Summing Waves

- Represent spatial signal as sum of sine waves (varying frequency and phase shift)
- Very commonly used to represent sound spectrum
Review: 1D Sampling and Reconstruction

- problems
  - jaggies – abrupt changes
  - lose data
Review: Sampling Theorem and Nyquist Rate

- Shannon Sampling Theorem
  - continuous signal can be completely recovered from its samples iff sampling rate greater than twice maximum frequency present in signal
- sample past Nyquist Rate to avoid aliasing
  - twice the highest frequency component in the image’s spectrum

![Image of sampling below the Nyquist rate](image-url)
Review: Aliasing

- incorrect appearance of high frequencies as low frequencies
- to avoid: **antialiasing**
  - supersample
    - sample at higher frequency
  - low pass filtering
    - remove high frequency function parts
    - aka prefiltering, band-limiting
Review: Low-Pass Filtering
Review: Picking Methods

- manual ray intersection
- bounding extents
- backbuffer coding
Review: Select/Hit Picking

- assign (hierarchical) integer key/name(s)
- small region around cursor as new viewport
  ![Diagram of small region around cursor]
  ![Diagram of new viewport]
- redraw in selection mode
  - equivalent to casting pick “tube”
  - store keys, depth for drawn objects in hit list
- examine hit list
  - usually use frontmost, but up to application