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

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?

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: 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

News

- homework 3 handed back, solutions out
- grades posted for P2, H2, H3
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
- 6-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
  \[(a_0, a_1, a_2) \times (b_0, b_1, b_2) = (a_0 b_0, a_1 b_1, a_2 b_2)\]

\[
\begin{pmatrix}
1, 1, 0.4 \\
0.7, 0.3, 1
\end{pmatrix}
\times
\begin{pmatrix}
0.5, 0.3, 0.8
\end{pmatrix}
= \begin{pmatrix}
0.5, 0.3, 0.8
\end{pmatrix}
\]

\[
\text{Light} \times \text{object} = \text{color}
\]

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?

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: 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

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

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: Gamma Correction
\[
\gamma_{DS} = \gamma_{D} \left( \frac{1}{\gamma_{DS}} \right)
\]
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 \(x, y\), slope \(0 < dy/dx < 1\)

Review: Bresenham Algorithm
- all integer arithmetic
- cumulative error function
- \(y=y0; e=0;\)
- for \((x=x0; x <= x1; x++)\) {
  - draw\(x, y\);
  - if \([2(e+dy) < dx]\) {
    - \(e = e+dy;\)
  - } else {
    - \(y=y+1;\)
    - \(e=e+dy-dx;\)
  - }
}

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

Review: General Polygon Rasterization
- idea: use a parity test
- for each scanline
  - \(edgeCnt = 0;\)
  - for each pixel on scanline (0 to r)
    - if (oldpixel->nextpixel crosses edge)
      - \(edgeCnt +=;\)
    - if \((edgeCnt & 1)\)
      - setPixel\(x\);
Interpolation

Review: Bilinear Interpolation
- interpolate quantity along $L$ and $R$ edges, as a function of $y$
  - then interpolate quantity as a function of $x$

\[ P(x, y) \]

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

\[ a_i = \frac{c_i}{d_i} \]

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>x=x&lt;min</th>
<th>x=x&lt;max</th>
<th>y=y&lt;min</th>
<th>y=y&lt;max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>0100</td>
<td>0101</td>
<td>0000</td>
</tr>
</tbody>
</table>

  - Trivial accept
  - \( \text{OC}(p1) = 0 \) \& \( \text{OC}(p2) = 0 \)
  - \( \text{OC}(p1) \) \& \( \text{OC}(p2) \)

  - Trivial reject
  - \( \text{OC}(p1) = 1010 \)
  - \( \text{OC}(p2) = 1001 \)

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

Review: 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

Visibility
**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: 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**
- 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: BSP Trees**
- Preprocess: create binary tree
  - Recursive spatial partition
  - Viewpoint independent

**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

Page 9
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

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
- \(\text{glTexCoord2d}(0,0)\);
- \(\text{glVertex3d}(-x, -y, -z)\);
- \(\text{glTexCoord2d}(1,1)\);
- \(\text{glVertex3d}(-x, y, z)\);
Review: Texture
- action when \( s \) or \( t \) is outside \([0...1]\) interval
  - tiling
  - clamping
- texture matrix stack
  ```
glMatrixMode( GL_TEXTURE );
```
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 \( p(x,y,z) \)

Procedural Approaches
Procedural Textures

- generate “image” on the fly, instead of loading from disk
- often saves space
- allows arbitrary level of detail


```python
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

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

Review: 1D Sampling and Reconstruction

- problems
  - jaggies – abrupt changes
  - lose data

Review: Sampling/Antialiasing

Sampling/Antialiasing

Review: Summing Waves

Review: Sampling Theorem and Nyquist Rate

- Shannon Sampling Theorem
  - continuous signal can be completely recovered from its samples if 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

Fig. 14.17 Sampling below the Nyquist rate. (Courtesy of George Wolberg, Columbia University)
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

Picking

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
- 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