Homework 4

- Q6 corrections, posted very late
- 8 bins, not 7
- for part b, give z-values in camera, not world, coordinate system
- hint on nonuniform depth was for camera, not except for 2-sided
- an anonymity system
- H4 solutions out soon for you to study, check web site. Sunday at latest. Contact me ASAP if you plan to turn in late, we will not accept late homeworks after solutions posted.

Reading from OpenGL Red Book

- 1: Introduction to OpenGL
- 2: State Management and Drawing Geometric Objects
- 3: Viewing
- 4: Display Lists
- 5: Color
- 6: Lighting
- 7: Texture Mapping
- 8: Selection and Feedback
- 9: Now That You Know
- only section Object Selection Using the Back Buffer
- Appendix: Basics of GLUT (Aux in v 1.1)
- Appendix: Homogeneous Coordinates and Transformation Matrices

Getting Help

- extra TA office hours in lab for hw/project
- Q&A
- Wed 2-4, Thu 4-6, Fri 9-6
- final review Q&A session
- Mon Apr 16 10-12
- reminder: my office hours Wed/Fri 11-12 in basement lab

Project 4 Grading

- project 4 grading slots signup
- Wed Apr 18 10-12
- Wed Apr 18 4-6
- Fri Apr 20 10-1

Final

- Tue Apr 17 8:30am-11:30am
- exam will be timed for 2.5 hours, but reserve entire 3-hour block of time just in case
- closed book, closed notes
- except for 2-sided 8.5”x11” sheet of handwritten notes
- fine to staple midterm sheets back to back
- calculators ok
- IDs out and face up

Final Emphasis

- covers entire course
- includes material from both midterms
- more than 1/3 on material after last midterm
- clipping
- hidden surfaces
- textures
- procedural approaches
- picking
- collision
- antialiasing
- visualization
- modern hardware
- curves

Review: Rendering Pipeline

- pipeline processing, set state as needed
  void display(){
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    Vertex3f(0.25, 0.25, -0.5);
    glEnd();
    glFlush();
  }

Reading

- forgot to post FCG Section 3.4, The Alpha Channel, as reading for Blending last Monday

Review: OpenGL

- do problems!
- work through old homeworks, exams

Reading from Shirley: Foundations of CG

1: Intro
- 10: Ray Tracing
2: Math
- 10.1-10.7, 10.9, 10.11.1
3: Linear Algebra
- 11: Texture Mapping
4: Rasterization
- 12: Graphics Pipeline
5: GPU
- only 10.12.4
6: Transformations
- 13: Data Structures
7: Viewing
- 15: Curves and Surfaces
8: Hidden Surfaces
- 17: Hardware
9: Surface Shading
- 18: Color
10: Shading
- 21: Visual Perception
11: Geometry
- only 21.2.2 and 21.2.4
12: Surface
- 25: Image-Based Rendering
13: Curves
- 26: Visualization
14: Surfaces

Review: Rendering Pipeline

- pipeline processing, set state as needed
  void display(){
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glDepthTest();
    glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
    glEnd();
    glFlush();
  }

Review – Fast!!

www.sgi.com/education/materials/hyperGraph/shuttle.htm
Review: Event-Driven Programming
- main loop not under your control
  - vs. procedural
- control flow through event callbacks
  - redraw the window now
  - key was pressed
  - mouse moved
- callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing

Review: 2D Rotation
- shear along x axis
  - push points to right in proportion to height
- reflect across x axis
  - mirror

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  1 & k & 0 \\
  0 & 1 & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  \cos(\theta) & -\sin(\theta) & 0 \\
  \sin(\theta) & \cos(\theta) & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

Review: 2D Rotation From Trig Identities
- Triad Identity:
  \[
  x' = r \cos(\phi) + x \\
  y' = r \sin(\phi) + y
  \]
  \[
  x = r \cos(\phi) - x' \\
  y = r \sin(\phi) - y'
  \]

Review: 2D Rotation: Another Derivation
- \[
  x' = x \cos(\theta) - y \sin(\theta) \\
  y' = x \sin(\theta) + y \cos(\theta)
  \]

Review: Shear, Reflection
- x-shear usually means shear along x in direction of some other axis
- affine transforms are combinations of
  - linear transformations
    - shear
    - scale
    - rotate
  - translations
    - projection
    - mirror

Review: Linear Transformations
- linear transformations are combinations of
  - properties of linear transformations
    - origin maps to origin
    - lines map to lines
    - ratios remain parallel
  - affine transformations
    - origin does not necessarily map to origin
    - parallel lines remain parallel
    - ratios are preserved
    - closed under composition

Review: Affine Transformations
- affine transforms are combinations of
  - linear transformations
  - translations
- properties of affine transformations
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Final Correction: 3D Shear
- general shear
  - \[
  \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix}
  \]
- “s-shear” usually means shear along x in direction of some other axis
- correction: need shear along same axis in direction of \(\alpha\)
- to avoid ambiguity, always say “shear along \(\alpha\) in direction of \(\alpha\)’s”

Review: Matrix Stacks
- OpenGL matrix calls postmultiply matrix \(M\) onto current
  - same relative position between object and basis vectors
- \(T_a \cdot T_b = T_b \cdot T_a\)

Review: Composing Transformations
- \(p' = T p\)
  - which direction to read?
    - right to left
      - interpret operations wrt fixed coordinates
      - moving object
    - left to right
      - interpret operations wrt local coordinates
      - changing coordinate system
      - OpenGL transforms current matrix with
        - \(glTranslate\(x, y, z\)\)
        - \(glRotate\(\phi, 0, 0, 1\)\)
        - \(glScale\(x, y, z\)\)
    - specify vector last, in final coordinate system
    - first matrix to affect is specified second-to-last

Review: Interpreting Transformations
- \(p' = T p\)
  - right to left: moving object
  - left to right: changing coordinate system

Review: Arbitrary Rotation
- arbitrary rotation: change of basis
  - \(A, B, C\) are orthogonal coordinate systems
    - \(A\) is location in \(X, Y, Z\) coordinate system
      - \(X = (x, y, z)\)
      - \(A = \begin{bmatrix}
  a_1 & b_1 & c_1 \\
  a_2 & b_2 & c_2 \\
  a_3 & b_3 & c_3
\end{bmatrix}
  \]
      - transformation from one to the other is matrix \(R\) whose columns are \(A, B, C\)
  - \(R(X) = \begin{bmatrix}
  a_1 & b_1 & c_1 \\
  a_2 & b_2 & c_2 \\
  a_3 & b_3 & c_3
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
  \]

Review: Transformation Hierarchies
- transforms apply to graph nodes beneath them
- design structure so that object doesn’t fall apart
- instancing

Review: Matrix Stacks
- OpenGL matrix calls postmultiply matrix \(M\) onto current
  - can save intermediate states with stack
  - no need to compute inverse matrices all the time
  - modular changes to pipeline state
  - avoids accumulation of numerical errors
Review: Display Lists
- precompile/cache block of OpenGL code for reuse
  - usually more efficient than immediate mode
  - exact optimizations depend on driver
  - good for multiple instances of same object
  - but cannot change contents, not parametrizable
  - good for static objects redrawn often
  - display lists persist across multiple frames
  - interactive graphics: objects redrawn every frame from
  - new viewpoint from moving camera
  - can be nested hierarchically
  - snowman example: 3x performance improvement, 36K polys

Review: World to View Coordinates
- translate eye to origin
- rotate view vector (lookat – eye) to w axis
- rotate around w to bring up into vw-plane

Review: Moving Camera or World?
- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
    - Note OpenGL models viewing in left-hand coordinates
  - camera stays put, but world moves to -7
  - resulting image same either way
  - possible difference: are lights specified in world or view
    coordinates?

Review: Orthographic Cameras
- center of projection at infinity
- no perspective convergence
- just throw away z values

Review: Transforming View Volumes
- scale, translate, reflect for new coord sys

Review: Basic Perspective Projection
- similar triangles

Review: Orthographic Derivation
- scale, translate, reflect for new coord sys

Review: Separate Warp From Homogenization
- warp requires only standard matrix multiply
- distort such that orthographic projection of
distorted objects is desired persp projection!
- w is changed
- clip after warp, before divide
- division by w: homogenization
Review: RGB Component Color
- simple model of color using RGB triples
- component-wise multiplication
  $$(a_0, a_1, a_2) \times (b_0, b_1, b_2) = (a_0 \times b_0, a_1 \times b_1, a_2 \times b_2)$$
- why does this work?
  - must dive into light, human vision, color spaces

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

Review: Projection Taxonomy
- perspective: projectors converge
  - orthogonal, axonometric
  - projectors parallel and perpendicular to projection plane
- oblique: projectors parallel, but not perpendicular to projection plane

Review: YIQ Color Space
- color model used for color TV
- Y is luminance (same as CIE)
- I & Q are color (not same I as HSV)
  - I represents color components
  - Q represents color components

Review: Open GL Example
  ```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45,1.0,0.1,200.0);
  glutSolidTeapot(1);
  glTranslate(4,4,0);
  glPushMatrix();
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  ```

Review: Chromaticity Diagram and Gamuts
- plane of equal brightness showing chromaticity
- gamut is polygon, device primaries at corners
  - defines reproducible color range

Review: Bresenham’s Line Rasterization Algorithm
- use error term, integer only
  ```
  // Bresenham's line algorithm
  void drawLine(int x0, int y0, int x1, int y1) {
    int dx = x1 - x0, dy = y1 - y0;
    int d = 2 * dy - dx, yerr = 2 * dy, xv = 1, yv = 0;
    if (dx < 0) {dx = -dx; y0 = -y0; yv = -yv;}
    if (dy > dx) {dx = dy; yv = dx;}
    int signD = dx < 0 ? -1 : 1;
    if (yerr > 0) {yerr -= dx; yv = 1;}
    while (x0 != x1) {
      if (yerr < 0) {yerr += dx; yv = 0;}
      else {yerr -= dx + 2 * dy; yv = 1;}
      y0 += yv;
      x0 += xv;
      drawPixel(x0, y0);
    }
  }
  ```
**Review: Specifying Normals**
- OpenGL state machine
  - uses last normal specified
  - if no normals specified, assumes all identical
- per-vertex normals
  - gNorm(1,1,1)
  - gNorm(0,0,0)
- per-face normals
  - gNorm(1,1,1)
  - gNorm(0,0,0)
- normal interpreted as direction from vertex location can automatically normalize (computational cost)
  - glNormal3f(GL_NORMALIZE);

**Review: OpenGL state machine**
- uses last normal specified
- if no normals specified, assumes all identical
- optimization: only loop over axis-aligned bounding box of xmin/xmax, yminymax

**Review: Light Sources**
- directional/parallel lights
  - point at infinity: (x,y,z,0)T
- point lights
  - finite position: (x,y,z,1)T
- spotlights
  - position, direction, angle
- ambient lights

**Review: Light Source Placement**
- geometry: positions and directions
- standard: world coordinate system
  - effect: lights fixed wrt world geometry
- alternative: camera coordinate system
  - effect: lights attached to camera (car headlights)

**Review: Shading Models**
- flat shading
  - compute Phong lighting once for entire polygon
- Gouraud shading
  - compute Phong lighting at vertices and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon

**Review: Recursive Ray Tracing**
- ray tracing can handle
  - reflection (chrome/mirror)
  - refraction (glass)
  - shadows
- one primary ray per pixel
- 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
  - termination criteria
    - no intersection (ray exits scene)
    - max bounces (recursion depth)
    - attenuated below threshold

**Review: Reflection and Refraction**
- refraction: mirror effects
  - perfect specular reflection
- refraction: at boundary
  - Snell's Law
    - light ray bends based on refractive indices $c_1, c_2$
    - $c_1 \sin \theta_1 = c_2 \sin \theta_2$
- Issues
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so don't have to test intersection with every object
Review: Image-Based Modelling / Rendering
- store and access only pixels
- no geometry, no light simulation, ...
- input: camera parameters, viewpoint
- divide surfaces into small patches
- loop: check for light exchange between all pairs
  - form factor: orientation of one patch w.r.t. other patch (n.o.n matrix)

Review: Subsurface Scattering
- light enters and leaves at different locations on the surface
- bounces around inside
- technical Academy Award, 2003
- Jensen, Marschner, Hanrahan

Review: Radiosity
[IBM] [IBM]
- capture indirect diffuse-diffuse light exchange
- model light transport as flow on a continuous surface
- input: set of images
- output: image from new viewpoint
- computational photography

Review: Non-Photorealistic Rendering
- simulate look of hand-drawn sketches or paintings, using digital models
- cool-to-warm shading: \( k = \frac{1 + n \cdot e}{2} \) \( k = 1 - k_1 \cdot e \)
- draw silhouettes: if \( (e \cdot n_h - n_e) = 0 \), \( e \) = edge-eye vector
- draw creases: if \( n_h \cdot n_e = \) threshold

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 \( 0 \)
- 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: BSP Trees
- preprocess: create binary tree
- recursive spatial partition
- viewpoint independent
- runtime: correctly traversing this tree enumerates objects from back to front
- viewpoint dependent: check which side of plane viewpoint is on at each node
- draw far, draw object in question, draw near

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: Cohen-Sutherland Line Clipping
- outcodes
- 4 flags encoding position of a point relative to top, bottom, left, and right boundary
- \( OC(p1) \) = 0 && \( OC(p2) \) = 0:
  - trivial accept
- \( OC(p1) \) & \( OC(p2) \) = 0:
  - trivial reject

Review: Sutherland-Hodgeman Clipping
- for each polygon vertex
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?

Review: Subsurface Scattering
- light enters and leaves at different locations on the surface
- bounces around inside
- technical Academy Award, 2003
- Jensen, Marschner, Hanrahan

Review: Sutherland-Hodgeman Clipping
- edge from \( p[i-1] \) to \( p[i] \) has four cases
- decide what to add to output vertex list

Review: Back-face Culling
- 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
- runtime: correctly traversing this tree enumerates objects from back to front
- viewpoint dependent: check which side of plane viewpoint is on at each node
- draw far, draw object in question, draw near

Review: Clipping
- analytically calculating the portions of primitives within the viewpoint

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: 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
- runtime: correctly traversing this tree enumerates objects from back to front
- viewpoint dependent: check which side of plane viewpoint is on at each node
- draw far, draw object in question, draw near

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

Review: Sutherland-Hodgeman Clipping
- 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
- 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: 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: Texture Coordinates
- texture image: 2D array of color values (texels)
  - assigning texture coordinates (s,t) at vertex with object coordinates (x,y,z,w)
  - use interpolated (4,1) for texel lookup at each pixel
  - use value to modify a polygon’s color
    - or other surface property
  - specified by programmer or artist

Review: Fractional Texture Coordinates

Review: Texture
- action when s or t is outside [0…1] interval
  - tiling
  - clamping
  - functions
    - replace/decal
    - modulate
    - blend
  - texture matrix stack

Review: Tiled Texture Map

Review: Fractional Texture Coordinates

Review: Environment Mapping
- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture
  - sphere mapping: texture is distorted fisheye view
  - point camera at mirrored sphere
  - use spherical texture coordinates

Review: MIPmapping
- image pyramid, precompute averaged versions

- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes

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

Review: Fractal Terrain
- 1D: midpoint displacement
  - divide in half, randomly displace
  - scale variance by half

Review: Fractal Terrain
- 2D: diamond-square
  - generate new value at midpoint
  - average corner values + random displacement
  - scale variance by half each time

Review: Language-Based Generation
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    - F = FLFRRFLF
  - Manzano’s Bush:
    - F = FF-[F+F+F+F][+F-F-F]
      - angle 16

Review: Particle Systems
- life cycle
  - generation, dynamics, death

Review: Particle Systems
- rendering tricks
  - avoid hidden surface computations

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: Reconstruction

Review: Bump Mapping: Normals As Texture
- create illusion of complex geometry model
  - control shape effect by locally perturbing surface normal

Review: Bump Mapping: Normals As Texture

Review: Environment Mapping

Review: Environment Mapping

Review: Visible 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: Invisible Primitives

Review: Texture

Review: Texture

Review: Texture

Review: Texture Coordinates

Review: Texture Coordinates

Review: Texture Coordinates

Review: Tiled Texture Map

Review: MIPmapping
**Review: Sub-Dividing Bézier Curves**
- find the midpoint of the line joining $M_{012}, M_{123}$, call it $M_{0123}$

**Review: de Casteljau's Algorithm**
- can find the point on Bézier curve for any parameter value $t$ with similar algorithm
  - for $t=0.25$, instead of taking midpoints take points 0.25 of the way

**Review: Transfer Functions To Classify**
- map data value to color and opacity
- can be difficult, unintuitive, and slow

**Review: Compositing**
- specify opacity with alpha channel: RGBA
  - $A = 1$: opaque, $A = 0$: transparent, $A = 0$: transparent
  - premultiply by alpha
    - $C = (1-A)B, A' = A$
    - $C' = B' + A' \cdot nB'$
    - $\gamma = \beta + n \cdot nB$
  - 1 multiply to find $C$, same equations for alpha and RGB

**Review: Vertex Shaders**
- replace model/view transformation, lighting, perspective projection
- a little assembly-style program is executed on every individual vertex independently
  - it sees:
    - vertex attributes that change per vertex:
      - position, color, texture coordinates...
    - registers that are constant for all vertices (changes are expensive):
      - matrices, light position and color, ...
    - temporary registers
    - output registers for position, color, tex coords...

**Review: Fragment Shaders**
- fragment shaders operate on fragments in place of texturing hardware
  - after rasterization
  - before any fragment tests or blending
  - input: fragment, with screen position, depth, color, and set of texture coordinates
  - access to textures, some constant data, registers
  - compute RGBA values for fragment, and depth
  - can also kill a fragment (throw it away)

**Review: Basis Functions**
- point on curve obtained by multiplying each control point by some basis function and summing

**Review: Comparing Hermite and Bézier**
- four control points, two of which are knots
- more intuitive definition than derivatives
- curve will always remain within convex hull (bounding region) defined by control points

**Review: GPGPU Programming**
- General Purpose GPU
- use graphics card as SIMD parallel processor
- textures as arrays
- computation: render large quadrilateral
- multiple rendering passes

**Review: Rendering Pipeline**
- so far rendering pipeline as a specific set of stages with fixed functionality
- modern graphics hardware more flexible
- programmable “vertex shaders” replace several geometry processing stages
- programmable “fragment/pixel shaders” replace texture mapping stage
- hardware with these features now called Graphics Processing Unit (GPU)
- program shading hardware with assembly language analog, or high level shading language

**Review: B-Spline**
- $C_0$, $C_1$, and $C_2$ continuous
- piecewise: locality of control point influence
Beyond 314: Other Graphics Courses

- 424: Geometric Modelling
  - was offered this year
- 426: Computer Animation
  - will be offered next year
- 514: Image-Based Rendering - Heidrich
- 526: Algorithmic Animation - van de Panne
- 533A: Digital Geometry - Sheffer
- 533B: Animation Physics - Bridson
- 533C: Information Visualization - Munzner