This will be a 75 minute written test covering everything we’ve done. Closed book, no notes or calculators. You can bring colored pipe cleaners if you wish.

It will include the basic ideas of the material on hierarchical objects but not the details of the `CS336Object` type. Some topics and review questions are listed below. The written homework from the beginning of the course provides some good examples for exam questions on transformations. Some other ideas for review or exam questions are shown in italics.

You may need to write short pieces of Javascript code or GLSL code. You do not have to memorize the details of specific OpenGL functions but you should be familiar with how to use them in loading data and rendering. (We will not worry about code to load and compile shaders). You’ll need to be able to write shader code at roughly the level of detail of the examples we’ve done and should be familiar with the basic GLSL types and operations (e.g. `mat4`, `vec3`, the dot and normalize functions, etc.) The table below, showing sample notation for matrices, will be given on the exam:

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>Javascript</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = Translate(dx, dy, dz)</td>
<td><code>M = new Matrix4().setTranslate(dx, dy, dz);</code></td>
</tr>
<tr>
<td>M = Scale(sx, sy, sz)</td>
<td><code>M = new Matrix4().setScale(sx, sy, sz);</code></td>
</tr>
<tr>
<td>M = RotateX(degrees)</td>
<td><code>M = new Matrix4().setRotate(degrees, 1, 0, 0);</code></td>
</tr>
<tr>
<td>M = RotateY(degrees)</td>
<td><code>M = new Matrix4().setRotate(degrees, 0, 1, 0);</code></td>
</tr>
<tr>
<td>M = RotateZ(degrees)</td>
<td><code>M = new Matrix4().setRotate(degrees, 0, 0, 1);</code></td>
</tr>
<tr>
<td>A = AB</td>
<td>A.multiply(B);</td>
</tr>
<tr>
<td>A = BA</td>
<td><code>A = new Matrix4(B).multiply(A);</code></td>
</tr>
<tr>
<td>C = AB</td>
<td>C = new Matrix4(A).multiply(B);</td>
</tr>
<tr>
<td>M = Translate(1, 2, 3) * Scale(4, 5, 6)</td>
<td><code>M = new Matrix4().setTranslate(1,2,3).scale(4,5,6);</code></td>
</tr>
</tbody>
</table>

References

Keep in mind that 100% of what we’ve covered has been applied in examples or homework, so you can always see it in action by studying the code!

See the resources page for textbook links,
http://web.cs.iastate.edu/~smkautz/cs336f15/resources.html

- Gortler Chapter 1 is a brief overview of the graphics pipeline.
• Teal Book Ch. 3, the first section ("Drawing Multiple Points"), includes some nice illustrations of OpenGL buffers. Also our first code example, GL_example1, includes comments explaining what's going on in detail.

• Gortler Chapters 2 - 4 cover transformation matrices and frames (though not the view and projection matrices) from a mathematical point of view.

• Teal Book chapters 3 and 4 talk about transformations at a more elementary level. Also see all the examples in the transformations directory, especially Transformations2.html.

• In addition, Teal Book Ch. 9 discusses the idea of hierarchical objects (the stuff explained in mathematical terms in Gortler Ch. 5). Also see the examples in the hierarchy directory.

• For the view and projection matrices, see Chapter 7 of the Teal Book. The section "Specifying the Viewing Direction" explains the view point and setLookAt function. The section "Specifying the Visible Range Using a Quadrangular Pyramid" has a nice diagram showing the parameters of a perspective projection using setPerspective or setFrustum. (Also see the comments in the definitions of the view and projection variables in transformations/RotatingCube.js, which you can play with.)

• The Teal Book Chapter 8 provides a good explanation of diffuse lighting and normal vectors, but does not discuss the specular component. This page, http://pages.cpsc.ucalgary.ca/~eharris/past/cpsc453/f10/tut20/ has a good picture and a brief explanation of the Phong reflection model. (Remember the symbols k and L stand for vectors with red, blue, and green component). However, the best way to understand the lighting model is to study the shader code in the examples in the lighting directory.

Topics and sample questions

The basic graphics pipeline:
   (Model -> )
   Vertex processing -> Primitive assembly -> Rasterization -> Fragment processing
   (-> framebuffer)

Role of shaders in the pipeline
   - Could you write a vertex shader that uses three adjacent vertices to calculate a normal vector? Explain briefly.
   - Could you write a shader that changes the way clipping is done?

OpenGL buffers and vertex attributes, role of function vertexAttribPointer
   - Explain what `gl.bindBuffer` does.
   - Describe the process of loading data and connecting data to vertex attributes
   - Could the same data be used by more than one shader without reloading the data on the GPU? Explain briefly.
- Given a sample of code such as our original GL_example1.js, modify it so that each vertex of the square has a different color and those colors are interpolated when it is rendered.

Linear interpolation
- Suppose you need to scale and bias the range 200 to 400 into the range -5 to 5. Write a formula to do it.
- Create an affine transformation $M$ such that given a coordinate vector $c = [x, y, 0, 1]^T$, $Mc = [x', y', 0, 1]^T$, where $x'$ is equal to $x$ converted from Celsius to Fahrenheit and $y'$ is $y$ converted from Fahrenheit to Celsius. (Write your answer as a the product of a scale matrix followed by a translation matrix.)

Basic color representation
Clip coordinates, clipping volume, depth testing
- Why does clipping potentially create new polygons?
- Describe the z-buffer algorithm in pseudocode

Vertex attributes, “varying” variables
- What is the role of a variable declared as attribute?
- What is the role of a varying variable?
- Write a fragment shader that colors each pixel with a greyscale value obtained from the average of the red, green, and blue values of a varying variable called fColor.
- Write a complete shader program (vertex shader and fragment shader) that will darken pixels according to their depth in clip space. Assume that the vertex position is an attribute and color is a uniform variable, and that the position is given in clip coordinates. Points that are closest to the viewer ($gl\_FragCoord.z = 0$) should have the given color, and points farthest from the viewer should be 50% of the given color.
- Write a complete shader program (vertex shader and fragment shader) similar to the above, using the depth in eye space. (You'll need a varying variable for the eye space depth, and uniform variables for the camera's near and far clipping planes.)

OpenGL primitives
- Given a set of vertices, show what is rendered using $gl\_DrawArrays(GL\_LINES, ...)$, $gl\_DrawArrays(GL\_TRIANGLE_FAN, ...)$, etc.

Indexed rendering
- Given a bunch of vertices, write down the indices that would be used to render it using $gl\_DrawElements(GL\_TRIANGLES, ...)$, or b) as a wireframe using $gl\_DrawElements(GL\_LINES, ...)$.

Linear transformations as matrices: given a transformation $f$, what does $f$ do to the basis vectors? Then the matrix for $f$ is $M = [f(e1) \ f(e2) \ f(e3)]$.

- Suppose $e1$ and $e2$ are the standard basis vectors, and there is a linear transformation $f$ that takes $e1$ to $[5 \ 7 \ 0]^T$ and $e2$ to $[-1 \ -3 \ 0]^T$. What is $f([2 \ 3 \ 0]^T)$?

Using homogeneous coordinates to represent translations
An affine transformation is a linear transformation followed by a translation; affine matrices

- Given an affine matrix \( M \),
\[
M = \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
write \( M \) as a product \( TA \), where \( T \) is a translation and \( A \) is linear.

- Prove that the 4d matrix for an affine transformation always has \((0 0 0 1)\) in bottom row.

2D rotation, translation, scaling

- Given a 2D triangle with vertices \((0, 0)\), \((2, 0)\), \((2, 1)\), sketch the triangle after
  a) translation by \((4, 5)\) followed by rotation by 90 degrees
  b) rotation by 90 degrees followed by translation by \((4, 5)\)
  c) translation by \((4, 5)\) followed by scaling by 2 in the y direction

  (show the approximate new coordinates in each case, e.g., on graph paper)

- For each of a, b, c above, write down the transformation matrix (as a product of
  standard matrices)

- A 2D triangle with vertices \((2, -3)\), \((3, -3)\), \((3, -5)\), is transformed to \((2, 2)\), \((2, 3)\), \((6, 3)\).
  Find a matrix for the transformation.

A frame is a set of basis vectors, plus an origin
Coordinate systems used in OpenGL: model, world, eye/view, clip, NDC, window, viewport

- What coordinate system are the fragment coordinates in (e.g. the gl_FragCoord
  variable accessible in the fragment shader)? What are the ranges for the x, y, and z
  values?

- What's the difference between clip coordinates and NDC?

Points vs vectors in homogeneous coords

Standard matrices: Translate(), RotateX(), RotateY(), RotateZ(), Scale()

- Suppose an object is centered at \((p1, p2, p3)\) and we want to rotate it through angle \( r \)
  about its z axis. Write down the necessary matrix in terms of the standard matrices (e.g.
  Translate(), etc.) (Don’t multiply them out!)

Euler angles

- Suppose an object is centered at \((p1, p2, p3)\) and we want to rotate it through angle \( r \)
  about a line through its center, where the line is directed at angle theta from the positive
  y-axis and angle phi from the positive x-axis, as in spherical coordinates. Write down the
  necessary matrix in terms of the standard matrices.

- Suppose we have performed a rotation of 45 degrees about the y axis and also a
  rotation of 30 degrees about the x axis. If the model’s x-axis is still in the x-z plane,
  which rotation was done first? That is, did we multiply coordinates by RotateY(45) *
  RotateX(30), or RotateX(30) * RotateY(45) ?

- Consider the transformation matrix RotateZ(90)*RotateX(90). Fill in the blanks in two
different ways to get the same transformation:
Intrinsic vs. Extrinsic transformations

Inverting a product of standard transformations

- An affine transformation consists of scaling by (1, 2, 1), then a translation to (1, 2, 3).
Write down a matrix for the inverse of this transformation. (You don’t have to multiply out the result.)
- Suppose that the first three columns of \( M \) are orthonormal:
\[
M = \begin{bmatrix}
    a & b & c & 0 \\
    d & e & f & 0 \\
    g & h & i & 0 \\
    x & y & z & 1 \\
\end{bmatrix}
\]
Find the inverse of \( M \).

Changing frames, \( M \) vs \( M^{-1} \), the pattern \( BMB^{-1} \)

- Suppose your new frame has origin (5, 10, 0) and that its x-axis is given by the vector \([0 1 0]^T\) and its y-axis is given by \([-1 0 0]^T\) (and z axis still \([0 0 1]^T\) ) (i.e. this can be pictured as taking place in a 2D plane)
  a) Find the matrix \( A \) that transforms the original frame into the second one
  b) If a given point has coordinates \((1, 2, 0)\) in the original frame, what are the coordinates of the point with respect to this new frame? Draw a picture to verify.
  c) Find the matrix \( M \) that gives you the new coordinates of any point with respect to the new frame, e.g. \( M[1, 2, 0, 1]^T \) would be the answer to (b)
- An object is centered at (2, 3, 4). What transformation will make the object 5 times as big without moving its center?
- Suppose some frame \( F' \) is obtained from the world frame by matrix \( A \). What matrix will rotate an object 30 degrees ccw about the z-axis of frame \( F' \)?

The view transformation

- Suppose you are given coordinates for three orthonormal vectors \( x = [x_1 \ x_2 \ x_3 \ 0]^T, y = [y_1 \ y_2 \ y_3 \ 0]^T \) and \( z = [z_1 \ z_2 \ z_3 \ 0]^T \), plus a point \( p = [p_1 \ p_2 \ p_3 \ 1]^T \), that together describe a frame that you would like to use as your camera. Write down the view matrix.
- Write the pseudocode for the function \( \text{LookAt}(\text{eye}, \text{at}, \text{up}) \). Assume you have basic functions such as \( \text{RotateX}() \), \( \text{Translate}() \), etc., and have dot and cross product operations.

Orthographic projections, view volume

- Describe what the function \( \text{setOrtho}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{near}, \text{far}) \) does.
- Why do we have to specify near and far clipping planes?

Perspective projection

- What would the arguments to the \( \text{setFrustum}(...) \) function be in order to give you the same matrix as \( \text{setPerspective}(30, 1.5, 2, 100) \) ?
- Where does perspective division occur in the pipeline?

Roles of the model, view, and projection matrices
- How do these three matrices relate to some of the coordinate systems we have discussed (model, world, eye/view, clip, NDC, window, viewport)?
- Describe an example showing why we don't always pre-multiply them
  \[\text{projection} \times \text{view} \times \text{model}\]

Normal vectors
- Does a normal vector always have to be perpendicular to an associated triangle? Briefly explain.

Lambert shading
- Why do we have to normalize \(L\) and \(N\) in our lighting calculation?
- Why does the shader code have to take the maximum of 0 and \((L \cdot N)\)?

Phong (ADS) lighting model
- Suppose at a given vertex, \(L\) points to the light and \(N\) is the normal vector and \(V\) points to the view point. Assume all three are normalized. Sketch a picture and label it. Write down the three components of the lighting model as accurately as you can.
- What is the purpose of the exponent in the specular component? What changes if we make the exponent bigger?

Gouraud and Phong shading
- Briefly explain the difference between Gouraud and Phong shading
- Explain the nature of the artifacts that occur using Gouraud shading with a specular component
- Why do we have to normalize after interpolating vectors? Give a concrete example showing that interpolating between two vectors of length 1 does not necessarily give you a vector of length 1.
- We normally do lighting calculations in eye coordinates. What would be different/easier/harder if we wanted to do it in world coordinates?
- Given the vertex shader from Lighting2c.html, add a uniform variable representing the position of the camera, given in world coordinates. Then rewrite the shader so that the lighting calculation is done in world coordinates. Note: you can make the assumption that there is no nonuniform scaling, i.e., the given normal matrix is equal to \(\text{view} \times \text{model}\).
- Same as above, but assume you don't have a uniform variable for the camera position. (The real question is, how do you get the camera's world-coordinate position if it's not given as a uniform variable? Recall that the view matrix looks like \(R^{-1}T^{-1}\) where the columns of \(R\) are the camera's basis vectors and \(T\) is the translation to the camera's position; also recall that \(R\) is orthonormal. This makes it easy to recover \(T\) from the view matrix.)

Coding examples
a) An excerpt from the draw() function of a Javascript program is shown on the next page. It renders a model consisting of the unit cube as in the screenshot on the, using the
identity matrix as the model transformation. Make the modifications to the code so that it will instead render the scene on the right, where the bars of the “X” are 4 units long and one-half unit across. (Note, the function `makeNormalMatrixElements()` just returns a 3x3 normal matrix corresponding to the given model and view matrices.)

b) An excerpt from the draw() and main() functions of a Javascript program is shown on the page after next. It renders an animation of a colored cube orbiting counterclockwise about the y-axis, in the x-z plane, in a circle of radius 100. Make the modifications needed so that the cube also spins on its own y-axis 365 times per full orbit.

c) Same setup as (b). Modify the code so that a second cube, one-tenth the size of the first one, orbits clockwise around the first one in a radius of 20, and at three times the speed of the first one's orbit. (Neither cube should spin on its own axis.)

```
// bind the shader and set attributes
gl.useProgram(shader);
var positionIndex = gl.getAttribLocation(shader, 'a_Position');
var normalIndex = gl.getAttribLocation(shader, 'a_Normal');
gl.enableVertexAttribArray(positionIndex);
gl.enableVertexAttribArray(normalIndex);
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.vertexAttribPointer(positionIndex, 3, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
gl.vertexAttribPointer(normalIndex, 3, gl.FLOAT, false, 0, 0);

// identity transformation
var model = new Matrix4();
// set uniforms in shader for projection * view * model transformation
var loc = gl.getUniformLocation(shader, "model");
gl.uniformMatrix4fv(loc, false, model.elements);
loc = gl.getUniformLocation(shader, "view");
gl.uniformMatrix4fv(loc, false, view.elements);
```
loc = gl.getUniformLocation(shader, "projection");
gl.uniformMatrix4fv(loc, false, projection.elements);
loc = gl.getUniformLocation(shader, "normalMatrix");
gl.uniformMatrix3fv(loc, false, makeNormalMatrixElements(model, view));

// draw the cube
gl.drawArrays(gl.TRIANGLES, 0, 36);

Part (b) and (c), excerpt from main()

var degrees = 0;
var increment = 2;

modelMatrix = new Matrix4(); // assume 'modelMatrix' is a global variable

var animate = function() {
    draw();

    degrees += increment;

    // orbit of radius 100 in the x-z plane
    var x = 100 * Math.cos(degrees * Math.PI / 180.0);
    var z = 100 * Math.sin(degrees * Math.PI / 180);
    modelMatrix.setTranslate(x, 0, z);

    requestAnimationFrame(animate, canvas);
};
animate();

Part (b) and (c), excerpt from Javascript draw() function called each frame
(assume 'projection' and 'view' are global variables)

var transform =
    new Matrix4().multiply(projection).multiply(view).multiply(modelMatrix);
var transformLoc = gl.getUniformLocation(shader, "transform");
gl.uniformMatrix4fv(transformLoc, false, transform.elements);
gl.drawArrays(gl.TRIANGLES, 0, 36);