Com S 336
Fall 2020
Exam 1 information and review problems
Thursday, October 15, 11:00 - 12:10

General information

This will be a 70-minute, timed exam, administered on Canvas during our usual class hour. Open notes, open books, but no collaboration. You can do internet searches for information, but you must pledge that you will not solicit help from any other person or online forum.

> *(Note: If you have an accommodation letter from Student Accessibility Services I'll contact you separately.)*

Please do try out the sample exam to get a sense of the format of the exam and to see what the text editor in Canvas is like.

Exam topics

The exam covers everything we've done through roughly September 29, not including lighting, and not including anything directly about quaternions. Remember that the *topics page* http://web.cs.iastate.edu/~smkautz/cs336f20/topics.html provides a detailed blow-by-blow review of what we've covered, with links to code examples.

Some ideas for review or exam questions are written in italics in the discussion below. Also, the written homework 2 provides some good examples for math-type questions on transformations.

You will 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 or 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 provided on the exam:

Pseudocode	Javascript
M = Translate(dx, dy, dz)	`M = new THREE.Matrix4().makeTranslation(tx, ty, tz);`
M = Scale(sx, sy, sz)	`M = new THREE.Matrix4().makeScale(sx, sy, sz);`
M = RotateX(degrees)	`M = new THREE.Matrix4().makeRotationX(toRadians(degrees));`
M = RotateY(degrees)	`M = new THREE.Matrix4().makeRotationY(toRadians(degrees));`
M = RotateZ(degrees)	`M = new THREE.Matrix4().makeRotationZ(toRadians(degrees));`
A = AB	`A.multiply(B);`
A = BA	`A = new THREE.Matrix4().copy(B).multiply(A);` `A.premultiply(B);  // alternate form`
C = AB	`C = new THREE.Matrix4().copy(A).multiply(B);`
M = Translate(1, 2, 3) * Scale(4, 5, 6)	`M = new THREE.Matrix4().makeTranslation(1,2,3)` `  .multiply(new THREE.Matrix4().makeScale(4,5,6));`

General overview of 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 topics page for specific textbook sections or references for topics, and see resources page for textbook links.

- Gortler Chapter 1, as well as Teal Book Chapter 1, offer brief overviews, 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_example1a, 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 than Gortler. Also see all the examples in the transformations directory, especially Transformations2.html and Rotations.html.
- For the view and projection matrices, see Chapter 7 of the Teal Book. The section "Specifying the Viewing Direction" explains the view point and camera matrix. The section "Specifying the Visible Range Using a Quadrangular Pyramid" has a nice diagram showing the parameters of a perspective projection using makePerspective (the same function is referred to as setFrustum in the teal book's libraries). Also see the comments in the definitions of the **`view`** and **`projection`** variables in transformations/RotatingCube.js, which you can play with.

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_example1a (both js and html files), 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 shift 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 the product of a scale matrix followed by a translation matrix.)*

Basic color representation as r, g, b, a

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. You'll need an attribute variable for the vertex position (assume the position is given directly in clip coordinates) and a uniform variable for the color. 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, but using the depth in eye/camera space. In this case assume there are uniform variables for model, view, and projection transformations. You'll need to define 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, sketch what is rendered using glDrawArrays(GL_LINES, ...), glDrawArrays(GL_TRIANGLE_FAN, ...), etc.

Linear transformations as matrices: given a transformation f, determine what f does to the basis vectors; then the matrix for f is M = [f(e1) f(e2) f(e3) $\vec{o}$] (first three columns; since f is *linear*, the origin is not moved).

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

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

- 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.

- From above, let M' = AT. Write M' as a product T'A', 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) (that is, coordinate vector $[0\ 0\ 0\ 1]^T$*), (2, 0), and (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 new coordinates in each case, e.g., on graph paper)

- For each of a, b, c above, write down the transformation matrix (in pseudocode, 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

Points vs vectors in homogeneous coords

Coordinate systems used in OpenGL that we have discussed so far: **model, world, eye/camera, clip, NDC, window**

- 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 is the difference between clip coordinates and NDC?

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

Euler angles

*- 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 by 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:*

*Rotate___(90) * Rotate___(90)*

*Rotate___(90) * Rotate___(90)*

Intrinsic vs. Extrinsic transformations

- Suppose you want to rotate a model 30 degrees around the y axis and then roll it 90 degrees around it's own z-axis. Write down the matrix you would use (product of standard transformation matrices

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 can use pseudocode, you don't have to multiply out the result.)

- *Suppose that the first three columns of M are othornormal:*

$$M=\begin{bmatrix} a & d & g & x \\ b & e & h & y \\ c & f & i & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Find the inverse of M.

Changing frames, M vs M^{-1}, the pattern AMA^{-1}

- *Suppose your new frame has origin (5, 10, 0) and that its x-axis is given by the vector [0 1 0 0]^T and its y-axis is given by [-1 0 0 0]^T (and z axis still [0 0 1 0]^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 1]^T in the original frame, what are the coordinates of this 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 *should 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 $\mathcal{F}'$ is obtained from the world frame by matrix A. What matrix will rotate an object 30 degrees ccw about the z-axis of frame $\mathcal{F}'$?*
- *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!)*
- *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 rotations about the coordinate axes by the standard matrices.*

The view transformation

- *Suppose you are given coordinates for three orthonormal vectors* $\boldsymbol{x}$ = [x1 x2 x3 0]^T, $\boldsymbol{y}$ = [y1 y2 y3 0]^T *and* $\boldsymbol{z}$ = [z1 z2 z3 0]^T, *plus a point* $\boldsymbol{p}$ = [p1 p2 p3 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 LookAt(eye, at, up). (Assume you have basic functions such as RotateX(), Translate(), etc., and have a* transpose() *operation for matrices, and* dot() *and* cross() *operations for vectors.)*
- *Given elements of the view matrix V, write code to find the coordinates of the camera position. (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 relatively easy to recover T from the view matrix.)*

Orthographic projections, view volume

- Describe what the function makeOrthographic(left, right, bottom, top, near, far) does. Derive the first row of the matrix (in terms of the given arguments).

- Why do we have to specify near and far clipping planes?

Perspective projection

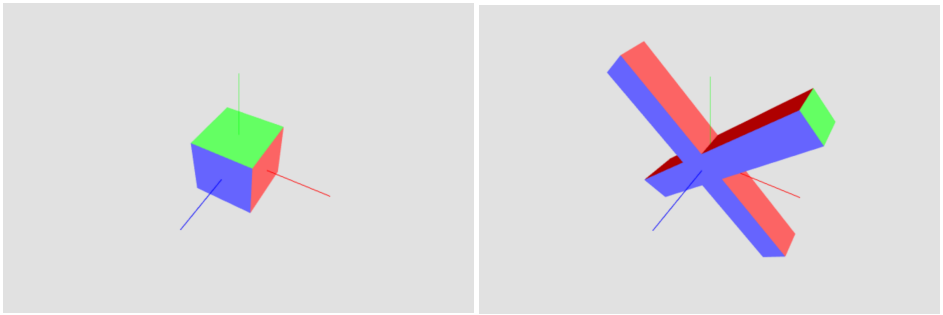
- What would the six arguments to the makePerspective(...) function be in order to give you a perspective projection matrix for a 30 field of view, aspect ratio 1.5, near plane 2, far plane 100 ? Write JS code.

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**)**?

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 left, using the identity matrix as the model transformation (the global variable **`modelMatrix`***). 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.*



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.)

Part (a), excerpt from Javascript draw() function called each frame. Assume view and projection are global variables that we do not need to modify:

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BIT);
gl.useProgram(shader);
var positionIndex = gl.getAttribLocation(shader, 'a_Position');
var colorIndex = gl.getAttribLocation(shader, 'a_Color');
gl.enableVertexAttribArray(positionIndex);
gl.enableVertexAttribArray(colorIndex);
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.vertexAttribPointer(positionIndex, 3, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
gl.vertexAttribPointer(colorIndex, 4, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, null);

var transform = new THREE.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);
```

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

```
var degrees = 0;
var increment = 2;

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

var animate = function() {
    draw();

    degrees += increment;

    // orbit of radius 100 in the x-z plane
    var x = 100 * Math.cos(degrees * Math.PI / 180);
    var z = 100 * Math.sin(degrees * Math.PI / 180);
    modelMatrix.makeTranslation(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 THREE.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);
```