Topics for Midterm Test

Framework

In lab [to be announced] with Safe Exam Browser.

Time limited [maximum scheduled time is lecture time].

All material covered in lectures, quizzes/worksheets, tutorials, and assignments; specific topics are listed here.

The test will be composed of:

- Short answer questions: multiple choice, true/false and fill-in-the-blank type questions.
- Computational problems that commonly arise in 2D graphics software development and rely solely on material covered this semester.
- Computational problems that could be categorized under what if? They aim to extend the knowledge base further; one way to do this is to amend the conventions used in class lectures.

Topics

1. Introduction to the graphics rendering pipeline. What is the purpose of each pipeline stage? What parts are programmable? What parts are fixed?
2. Introduction to the OpenGL/GLSL toolbox. What is OpenGL? What is OpenGL rendering context? What is the OpenGL extension mechanism? What is the purpose of GLFW? What is the purpose of GLEW? What is a framebuffer? What is double buffering? How are colors represented? What is GLSL? What are vertex attributes? How is geometry represented? What are structure of array and array of structure representation of vertices? What are OpenGL primitives? You should know about points, lines, triangles, triangle fans, and triangle strips. You should know about explicit and indexed representation of primitives. You should be able to perform an analysis of the memory storage and transfer requirements of indexed and explicit methods for triangles, fans, and strips. What are vertex buffer objects? What are vertex index objects? What is GLSL? What are vertex and fragment shaders? How do you pass vertex and geometry information to vertex shaders? How does the vertex shader transform input data? What happens to this transformed data? What does the fragment shader do? What happens to the data generated by the fragment shader? What is a shader program object? How is GLSL source code transformed into programs that get executed by GPU cores? Know the OpenGL draw calls associated with explicit and indexed vertex data.
3. 2D affine transforms:
    1. Matrix manifestations of vectors and points; right- and left-handed coordinate basis and reference frames;
    2. Computing determinants and inverses of 3×3 matrices.
    3. You should know: how programmers use (built-in) matrices in compatible OpenGL, GLSL, GLM; how to specify column-major matrices using one-dimensional arrays in row-major memory addressing programming languages such as C and C++.
    4. Geometrical meaning of affine transforms consisting of an unknown sequence of scales, rotations, and translation [not necessarily in that order]: you should know the meaning of *standard 2D box* [axis-aligned square having dimensions 2×2 with center at origin]; you should be able to geometrically interpret what happens when an affine transform is

applied to standard 2D box: after applying 3×3 matrix $\mathbf{M}$ representing concatenation of scales, rotations, and translations on basis vectors $\hat{i}$, $\hat{j}$, they're transformed to vectors specified by $\mathbf{M}$'s 1^{st} and 2^{nd} columns, respectively, the origin [standard 2D box's center] is transformed to the position specified by $\mathbf{M}$'s 3^{th} column. This is the geometrical meaning of an affine transformation and is the most critical and important thing discussed in class lectures.

5. Things you should know about scaling: uniform and non-uniform scaling and what they do to standard 2D box; meaning of inverse scale transform and its matrix manifestation; how to create a single matrix consisting of scaling transform followed by translation transform without matrix computations; how to scale about a pivot point [that is, the pivot point is invariant to the scale transform].
6. Things you should know about rotation: what is the rotation axis for 2D rotations; how to define counter-clockwise and clockwise angular displacements; derive rotation matrix; know that rotation matrices are orthogonal matrices; know properties of orthogonal matrices; know what rotation does to standard 2D box.

4. 2D Model transform or model-to-world transform: You should know how to create a model transformation matrix consisting of scaling followed by rotation followed by translation transform without resorting to matrix multiplications; how to transform vectors and points from world to model frame; you should what happens if the scale-rotation-translation order is changed to rotation followed by scale followed by translation and you should be able to compute model transformation matrix without resorting to matrix multiplications.
5. 2D View Transforms: You should know meanings of terms *view point*, *eye*, *camera*, *target*, *view vector*, *view frame*; you should know how to define an orthonormal right-handed view frame with camera looking along its $x-$ or $y-$ axis; how to compute a matrix that transforms vectors and points from world frame to view frame and vice versa; you should know meaning of *view transformation matrix*.
6. 2D NDC frame: You should know the purpose of NDC; meaning of *standard NDC 2D box*; path taken by points from model frame to get to NDC frame; convention used by OpenGL to define NDC; know viewport transformation that maps NDC frame point P^n to window frame point P^d; know how to derive a replacement for the OpenGL command `glViewport`.
7. 2D Texture mapping: Know the texture mapping pipeline; know about texture images and the memory layout of image data; know about texels and texel coordinate system; know about normalized texture coordinate system; know how to apply wrapping (tiling), mirroring, and clamping functions to texture coordinates; know how to map points between texel and texture coordinate systems; know what point sampling is; know what texture sampling in a fragment shader means; know how to map a texture coordinate to a specific memory.
8. Basic knowledge of GLSL: Know about the basic elements that have been discussed in terms of vertex and fragment shaders; what are `in`, `out`, and `uniform` variables; know how to use GLSL's vertex and matrix [`vec2`, `vec3`, `mat3` for example] formats; know what the `version` and `layout` specifiers do? Know the behavior of `uniform` variables? What happens if you don't update them every draw call?
9. Rendering wireframe images: What does scan conversion mean? what does rasterization mean? you should know the explicit or slope-intercept equation of a line; how is the line segment converted to pixels? how does the line's slope affect how the line is sampled? what is the naïve rasterization algorithm for line segment? how do you generalize the algorithm to work for line segments with different slopes and different sampling directions? how do you improve the naïve algorithm? what does the DDA algorithm do? what are the disadvantages of DDA algorithm? what is the Bresenham's integer-only algorithm? Given start point $P_1\left(x_1, y_1\right)$ and $P_2\left(x_2, y_2\right)$ and $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$, you should know how to derive for line with slopes $0 \leq m \leq 1$, or with slope $-1 \leq m < 0$, or with slope $m > 1$,

or with slope $m < -1$ and when $\Delta x < 0$ or $\Delta x \geq 0$ or when $\Delta y < 0$ or when $\Delta y \geq 0$. Remember since Bresenham's algorithm is integer-only, any floating-point values or results [including intermediate steps] will be considered wrong.

10. I've tried to be as comprehensive as possible - however being human I may have forgotten some details. In that case, if those details have been covered in class lectures, you should review that material as it may show up in the test.