

CS351-1 Introduction to Computer Graphics

Test A: Shape
100 pts max.

Feb 02, 2018

NetID jtt3923 Name Junda Tu

(netID is 6 letters + 6digits, e.g. jet861)

INSTRUCTIONS: Edit this file in Microsoft Word or in Google Docs to enter your **HIGHLIGHTED** answers. Upload your own file on Canvas before the end of the day Sunday, Feb. 04, 2018, 11:59PM.

Corrected typos on problems
6 & 15 shown in blue highlight

GUI and User Interface:

HIGHLIGHT your choice to mark your answer.

1) (5 pts) What are the dimensions of the 'Canonical View Volume' (CVV) in OpenGL and WebGL?

- A) Fixed—a unit cube centered at the origin whose x,y,z values span (+/-1, +/-1, +/-1)
- B) Fixed—a unit cube, with x,y,z origin shown at the upper-left of display: ($0 \leq x,y,z \leq 1$)
- C) Fixed—an on-screen rectangle whose limits vary and depend on the modelMatrix contents
- D) adjustable—x,y origin set at upper left, and x,y max values set by canvas width and height
- E) adjustable—a unit cube centered at the origin
whose visible x,y,z values span(+/- width/2, +/-height/2, +/-1)
- F) Something else; none of the above.

GPU Communication:

2) (24pts) TRUE/FALSE: (copy-and-paste your choice of these highlighted answers "True" or "False")

- a) False WebGL *requires* users to specify all vertex positions using real values (floats). This requirement ensures that limited precision won't introduces rendering flaws on-screen.
- b) False GLSL supplies a standard set of functions that can create a 4x4 matrix for translation, for rotation, or for scale, each from a single function call.
- c) False Drawing commands for WebGL and drawing commands for HTML5 'canvas' elements share the same on-screen drawing axes; both span +/- 1, with origin at center.
- d) True Of all the many parameters kept as state variables by WebGL, such as background color, buffer bindings, depth testing, etc. some can be read back from the GPU; others cannot.
- e) False A 'Fragment Shader' is optional; without it, your WebGL/HTML5/JavaScript program can still draw single-color WebGL drawing primitives on-screen (e.g. TRIANGLES).
- f) True With the proper selection of 'stride' and 'offset', WebGL can render the contents of a vertex buffer object (VBO) that holds 100 vertex positions, followed 100 vertex colors, followed by 100 vertex surface normals. In this VBO, the vertex attributes are NOT interleaved!
- g) False WebGL prevents use of the same 'uniform' variable to send values to both the Vertex Shader and Fragment Shader. If JavaScript sets its value, only one shader can use it.
- h) False WebGL itself provides built-in functions for mouse, keyboard, and window-system interactions. We use HTML and JavaScript functions instead because they are more convenient.

Vector-Matrix Math:

In a WebGL program of the sort developed and described in your textbook (e.g. starting with Week02 Vector Matrix Tests, Ch2, 'HelloMatrixOps.js', w/ cuon-matrix-quat.js library) we find two 'Vector4' objects (or 'variables') named 'aVec' and 'bVec', and one 'Matrix4' object named 'aMat' that holds this 4x4 matrix:

```
[a b c d]
[e f g h] == aMat
[j k m n]
[p q r s]
```

3) (4 pts) If we call `aMat.setTranslate()`, then apply `aMat` to transform `aVec` into `bVec` like this:

`bVec = aMat.multiplyVec4(aVec);` then the new value of `bVec.elements[1]` must be:

- A) `j*aVec.elements[0] + k*aVec.elements[1] + m*aVec.elements[2] + n*aVec.elements[3];`
- B) `b*aVec.elements[0] + f*aVec.elements[1] + k*aVec.elements[2] + q*aVec.elements[3];`
- C) `e*aVec.elements[0] + f*aVec.elements[1] + g*aVec.elements[2] + h*aVec.elements[3];`
- D) `c*aVec.elements[0] + g*aVec.elements[1] + m*aVec.elements[2] + r*aVec.elements[3];`
- E) something else happens; none of the above.

4) (4 pts) For that same translation matrix `aMat`, when `aVec.elements[3]` value is 1.0, then:

- A) All four elements of `bVec` result will always be 1.0, for any and all translations.
(e.g. `bVec.elements[0]=bVec.elements[1]=bVec.elements[2]=bVec.elements[3]=1`).
- B) Only `bVec.elements[0]` result will be 1.0; all other elements vary with translation amount.
- C) Only `bVec.elements[3]` result will be 1.0; all other elements vary with translation amount.
- D) For all non-zero translation distances, all of the elements of `bVec` will vary.
- E) Something else happens; none of the above.

5) (4 pts) For that same translation matrix `aMat`, when `aVec.elements[3]` value is 0.0, then:

- A) All four elements of `bVec` result must be 0.0 for any and all translation amounts.
(e.g. `bVec.elements[0]=bVec.elements[1]=bVec.elements[2]=bVec.elements[3]=0`).
- B) Only `bVec.elements[0]` result must be 0.0; all other elements vary with translation amount.
- C) Only `bVec.elements[3]` result must be 0.0; all other elements vary with translation amount.
- D) For all non-zero translation distances, all of the elements of `bVec` will vary.
- E) Something else happens; none of the above.

6) (4 pts) If we call `aMat.setScale()` then apply `aMat` to transform `aVec` into `bVec` like this:

`bVec = aMat.multiplyVec4(aVec);` and then find that `bVec.elements[1] == 0` despite an nonzero value for `aVec.elements[1]`, then we know something about the `aMat` matrix:

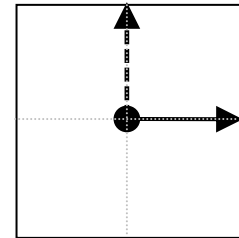
- A) Only these three `aMat` matrix elements are nonzero; 's', 'a', and 'm'
- B) Only these three `aMat` matrix elements are nonzero; 'm', 'f', and 's'
- C) Only these three `aMat` matrix elements are nonzero; 's', 'a', and 'f'
- D) Only these three `aMat` matrix elements are nonzero; 'm', 'f', and 'a'
- E) Something else happens; none of the above.

- 7) (4 pts) If we call `aMat.setRotate(-90,0,0,-1)`; then apply `aMat` to transform `aVec` into `bVec` :
`bVec = aMat.multiplyVec4(aVec)`; we can be certain that:
- A) `bVec.elements[0]` equals `aVec.elements[0]`
 - B) `bVec.elements[0]` equals `-aVec.elements[0]` (note minus sign)
 - C) `bVec.elements[0]` equals `aVec.elements[1]`
 - D) `bVec.elements[0]` equals `-aVec.elements[1]` (note minus sign)
 - E) `bVec.elements[0]` equals `aVec.elements[2]`
 - F) `bVec.elements[0]` equals `-aVec.elements[2]` (note minus sign)
 - G) Something else happens; none of the above.
- 8) (4 pts) For that same rotation matrix, when `aVec.elements[3]` value is 0.0, then:
- A) All four elements of `bVec` result will always be 0.0 for any and all rotations we specify.
 (e.g. `bVec.elements[0]=bVec.elements[1]=bVec.elements[2]=bVec.elements[3]=0`).
 - B) Only `bVec.elements[0]` result must be 0.0, and `aVec.elements[1] == bVec.elements[1]`
 - C) Only `bVec.elements[1]` result must be 0.0, and `aVec.elements[0] == bVec.elements[0]`
 - D) Only `bVec.elements[2]` result must be 0.0, and `aVec.elements[3] == bVec.elements[3]`
 - E) Only `bVec.elements[3]` result must be 0.0, and `aVec.elements[2] == bVec.elements[2]`
 - F) Something else happens; none of the above.

Matrix Duality & Scene Graphs

Suppose that:

- Our HTML5 Canvas is square on-screen (height==width), and it displays WebGL output.
- We wrote a Javascript `drawAxes()`; function that causes WebGL to:
 - draw a solid arrow from the origin to $(+1,0,0)$ to depict the x axis, and
 - draw a dashed arrow from the origin to $(0,+1,0)$ to depict the y axis.
 - draw a large, solid round 'dot' at the origin
- In JavaScript we send the 4x4 'modelMatrix' as a uniform to the GPU,
- Our Vertex shader applies that uniform matrix to transform all vertex position attributes before drawing them.
- This code: `modelMatrix.setIdentity(); drawAxes();`
- Causes our program to draw the picture shown, with both arrows drawn entirely within the canvas.



If our program executes this sequence of statements instead, smoothly varying $0 \leq \text{myAngle} \leq 90^\circ$

```
modelMatrix.setTranslate(-0.5,-0.5,-0.5); // 3D move
modelMatrix.scale(0.5, 1.0, 0.5);        // non-uniform scaling
modelMatrix.rotate(-myAngle,0.0,0.0,1.0); // rotate (animated)
drawAxes();                               // draw it!
```

then:

- 9) (4 pts) The central 'dot' for the arrows will be drawn:
- A) in the center of the upper right quarter (or 'quadrant') of the HTML-5 canvas;
 - B) in the center of the upper left quarter (or 'quadrant') of the HTML-5 canvas;
 - C) in the center of the lower right quarter (or 'quadrant') of the HTML-5 canvas;
 - D) in the center of the lower left quarter (or 'quadrant') of the HTML-5 canvas;
 - E) at the exact center of the Canvas (e.g. the same 'dot' location shown in the drawing above)
 - F) Something else happens; none of the above
- 10) (4 pts) As the 'myAngle' value varies smoothly between 0 and 90, where will we find the center of rotation (e.g. the 'hinge point') in the on-screen drawing?

- A) in the center of the upper right quarter (or 'quadrant') of the HTML-5 canvas;
- B) in the center of the upper left quarter (or 'quadrant') of the HTML-5 canvas;
- C) in the center of the lower right quarter (or 'quadrant') of the HTML-5 canvas;
- D) in the center of the lower left quarter (or 'quadrant') of the HTML-5 canvas;**
- E) at the exact center of the Canvas (e.g. the same 'dot' location shown in the drawing above)
- F) Something else happens; none of the above

11) (4 pts) The smoothly-changing 'myAngle' variable animates the drawing. As its value changes,

- A) all parts of both arrows stay entirely on-screen;**
- B) part of the solid arrow sometimes goes off-screen, but the dashed arrow is always fully visible;
- C) part of the dashed arrow sometimes goes off-screen, but the solid arrow is always fully visible;
- D) Parts of both arrows sometimes go off-screen, but not at the same time;
- E) Something else happens; none of the above

12) (4 pts) When 'myAngle' == 0, does the solid arrow have the same length as the dashed arrow on-screen?

- A) Yes: the solid-arrow is drawn on-screen with exactly the same length as the dashed arrow
- B) No: the solid-arrow is drawn on-screen with a longer length than the dashed arrow
- C) No; the solid-arrow is drawn on-screen with a shorter length than the dashed arrow**
- D) Something else happens; none of the above.

13) (4 pts) When 'myAngle' varies smoothly from 0 to 90, does solid-arrow length change on-screen?

- A) No; the solid arrow rotates on-screen but does not change its length**
- B) Yes; solid arrow changes length on-screen as it rotates, with maximum length at myAngle=0.
- C) Yes; solid arrow changes length on-screen as it rotates, with maximum length at myAngle=90.
- D) Something else happens; none of the above.

14) (4 pts) When 'myAngle' varies smoothly from 0 to 90, does dashed-arrow length change on-screen?

- A) No; the dashed arrow rotates on-screen but does not change its length**
- B) Yes; dashed arrow changes length on-screen as it rotates, with maximum length at myAngle=0.
- C) Yes; dashed arrow changes length on-screen as it rotates, with maximum length at myAngle=90.
- D) Something else happens; none of the above.

Draw your own 'scene-graph' for our set of 4 statements (listed above question 9).

15) (4 pts) The graph will contain a node for each of the 4 statements, and

- A) The transform node for **scale()** is a descendant of the 'transform' node for **rotate()**
- B) The transform node for **rotate()** is a descendant of the 'transform' node for **scale()****
- C) The **rotate()** and **scale()** nodes are siblings – both are child nodes of the same group node
- D) The **rotate()** and **scale()** nodes are unrelated—they do not share the same parent node
- E) Something else happens; none of the above.

16) (4 pts) Our sequence of 4 statements illustrates how to traverse a scene-graph to draw the animated, jointed objects it describes. In general, we traverse a scene-graph (not just ours) to generate statements:

- A) In breadth-first order, starting from the top (root) of the graph, ending at the leaf nodes
- B) In breadth-first order, starting with the leaf nodes, and always ending at the top (root)
- C) In depth-first order, starting from the top (root) of the graph, ending at the leaf nodes**
- D) In depth-first order, starting from each leaf node in turn, and always ending at the top (root).
- E) Something else happens; none of the above.

- 17) (5 pts) If we use [T] to represent the 4x4 matrix that performs our translation, and [R] to represent the 4x4 matrix that performs our rotation, and [S] to represent the 4x4 matrix for scale,

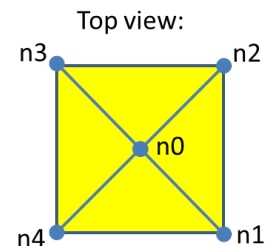
The modelMatrix variable contains a 4x4 matrix. Just before we call drawAxes(), the modelMatrix values are the result of combining several transformation matrices. How were they combined?

- A) By matrix multiply: $[\text{modelMatrix}] = [\text{T}][\text{R}][\text{S}]$
 (HINT: if you multiply a Vector4 by modelMatrix, we could get the same result if we multiplied the Vector4 first by S matrix, then R matrix, then T).
 B) By matrix multiply; $[\text{modelMatrix}] = [\text{T}][\text{S}][\text{R}]$
 C) By matrix multiply: $[\text{modelMatrix}] = [\text{R}][\text{T}][\text{S}]$
 D) By matrix multiply; $[\text{modelMatrix}] = [\text{R}][\text{S}][\text{T}]$
 E) By matrix multiply; $[\text{modelMatrix}] = [\text{S}][\text{T}][\text{R}]$
 F) By matrix multiply; $[\text{modelMatrix}] = [\text{S}][\text{R}][\text{T}]$
 G) Something else happens; none of the above.

Vertex Sequencing

- 18) (4 pts) If we use WebGL's gl.TRIANGLES drawing primitive (not TRIANGLE_STRIP), how many vertices do we need to draw a 3D cube?
- A) 8: one vertex per corner; however, this limits each corner to 1 color only (where 3 faces meet)
 B) 20: 8 corners + 6 faces*2 more vertices to split the face into 2 triangles
 C) 24: 8 corners * 3 vertices/corner (one vert for each face)
 D) 36: 6 faces * 2 triangles/face * 3 vertices/triangle.
 E) Something else happens; none of the above.

- 19) (6 pts) There are many ways to tessellate 4-sided pyramid with these nodes (square pyramid base is facing away from you) into one single triangle strip (gl.TRIANGLE_STRIP, not gl.TRIANGLES). A 'good' tessellation makes a strip that covers all surfaces with no redundancies, no degenerate triangles, and always-correct winding order, such as this sequence of vertex-locations:



- A) n2, n0, n3, n4, n2, n1, n0, n4
 B) n0, n2, n0, n3, n4, n2, n1, n0, n4
 C) n3, n0, n2, n1, n3, n4, n0, n1
 D) n2, n3, n0, n4, n1, n2, n0, n2, n4, n3
 E) Something else. None of these sequences tessellate the shape correctly.