Name _____ *Formatted For Two-Sided Printing*

Staple This Side

GPU Programming
EE 4702-1
Midterm Examination

Friday, 31 October 2025, 9:30-10:20 CDT

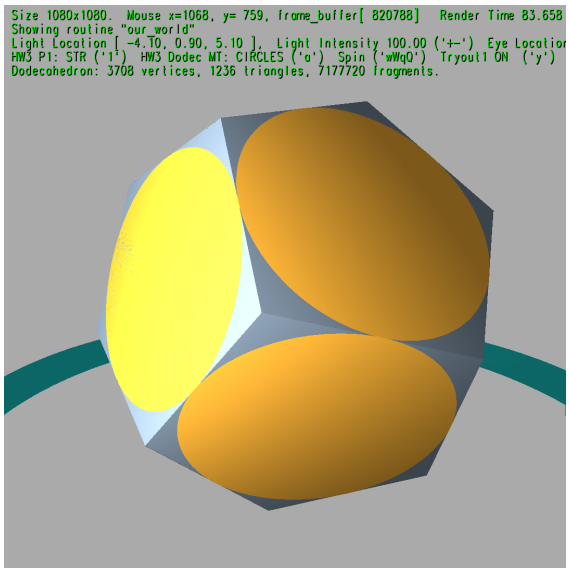
Problem 1	_____	(35 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(20 pts)
Problem 5	_____	(15 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Staple This Side

Problem 1: [35 pts] Appearing to the right is a dodecahedron with gold circles just above its faces. The code below, based on the Homework 3 solution, draws the dodecahedron but the circle code is incomplete. Complete it.



- ☐ Several quantities are computed for you, such as the pentagon center and the dodecahedron center.
- ☐ Approximate the circle as an `n_sides` sided polygon.
- ☐ Use `ri` for the radius of the circle.
- ☐ The circle plane (circle local `ax`,`ay`) should be parallel to the pentagon plane.
- ☐ Place the circle center on the line from the dodecahedron center, `dod_ctr`, to the pentagon center, `pen_ctr`, a very small distance away from pentagon. See `vcc`.
- ☐ Just emit circle coordinates, **don't emit colors**.

```
case MT_Circles: {
  pCoor dod_ctr = cyl_center + pVect(0,1.5,0);          // Dodecahedron Center
  float ri = len_edge / ( 2 * tanf(numbers::pi/5) );    // Circle Radius
  int n_sides = 100;                                    // Number of "circle" sides
  float delta_theta = 2 * numbers::pi / n_sides;

  for ( size_t i=0; i<coors.size(); i+=5 ) {
    for (int j=1; j<4; j++) coors_os<<coors[i]<<coors[i+j]<<coors[i+j+1]; // Emit pentagon.
    pVect vi(0,0,0); for ( int j=1; j<5; j++ ) vi += pVect(coors[i],coors[i+j]);

    // Dodecahedron Face (Pentagon) Coords in coors[i] to coors[i+4];
    pCoor pen_ctr = coors[i] + 0.2 * vi;                // Pentagon Center
    pVect vcc( dod_ctr, pen_ctr );                      // Should be useful.




    for ( int j =    ; j  <   n_sides;  j++ ) { //  ☐ Finish this line.
      float theta = j * delta_theta;




       coors_os <<
  }}
  gc.topology_set(Topology_Individual);
```


Staple This Side

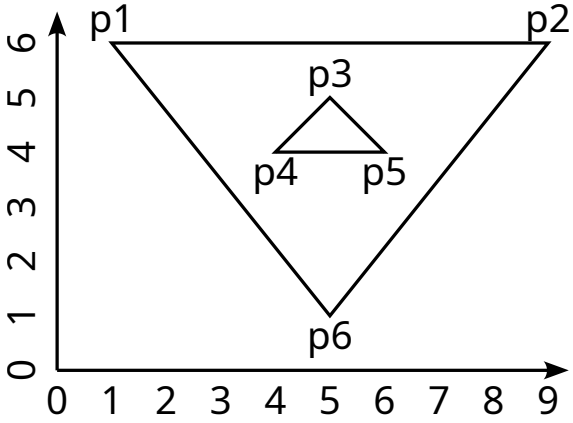
Staple This Side

Problem 2: [15 pts] Below are two unsuccessful attempts to render a triangle-in-a-triangle shape using the `Our_3D` demo code from class, part (c) is your shot at success.

(*a*) The code below inserts vertices for an **individual triangle topology**.

☐ On the illustration to the right show the triangles that **actually are** rendered. ☐ Label each triangle with a letter (a, b, c, ...).

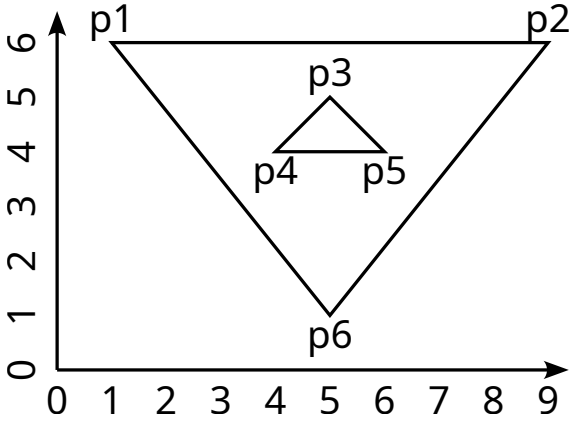
```
// Pipeline set up for individual triangles.
// (Vulkan: PrimitiveTopology::eTriangleList).

coors_os << p1 << p2 << p3 << p4 << p5 << p6;
```



(*b*) The code below inserts vertices for a **triangle strip topology**.

☐ On the illustration to the right show the triangles that **actually are** rendered. ☐ Label each triangle with a letter (a, b, c, ...).

```
// Pipeline set up for a triangles strip.
// (Vulkan: PrimitiveTopology::eTriangleStrip).

coors_os << p1 << p2 << p3 << p4 << p5 << p6;
```

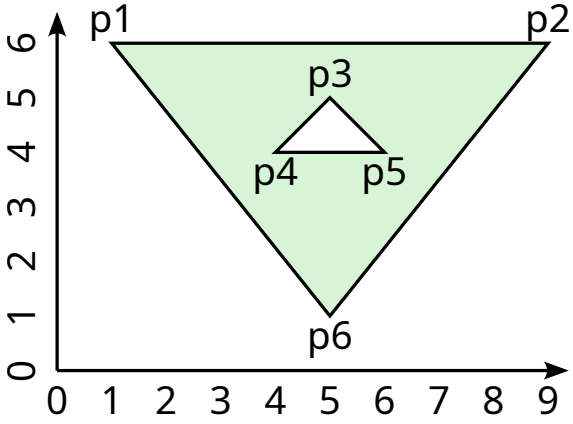


(*c*) The code below is incomplete.

☐ Complete the code below so that green shaded area shown is rendered correctly, ☐ and without overlapping triangles, for a pipeline set for a triangle strip topology.

☐ Coordinates `p1` through `p6` are already defined correctly, don't assign values to them.

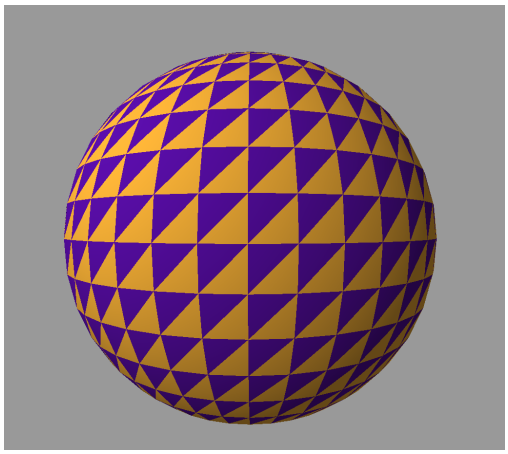
```
// Pipeline set up for a triangles strip.
// (PrimitiveTopology::eTriangleStrip).

coors_os <<
```



Staple This Side

Problem 3: [15 pts] Answer the following questions about spheres rendered using rasterization.

(a) The sphere shown to the right consists of $T = 1600$ triangles. (Not all are visible.)



Original Sphere

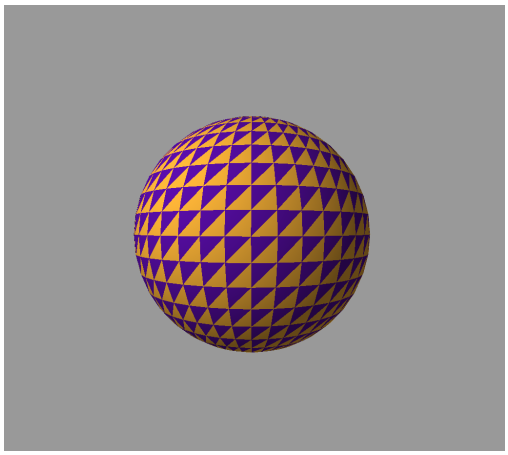
☐ How many vertices would be needed to render the sphere using a individual triangles (a triangle list)? ☐ Answer in terms of T.

☐ How many vertices would be needed to render the sphere using a triangle strip? ☐ Answer in terms of T.

To render the original sphere using a triangle list about $F = 500,000$ fragments were generated.

☐ About how many fragments would be generated if the sphere were rendered using a triangle strip? ☐ Answer in terms of F.

(b) Suppose for the original sphere above the eye were at distance d from the nearest point on the surface of the sphere. Let V_o denote the number of vertices used to render the original sphere using triangle strips and F_o denote the number of fragments generated rendering the original sphere. In the otherwise identical *far sphere* to the right the eye is at distance $2d$ from the nearest point. Both the original and far spheres have T triangles. *Note: In the original exam F_o was not mentioned, and it was not clear whether V should denote the number of vertices in the near or far sphere.*



Far Sphere

☐ How many vertices in the far sphere? ☐ Answer in terms of T, V_o, F_o, and d.

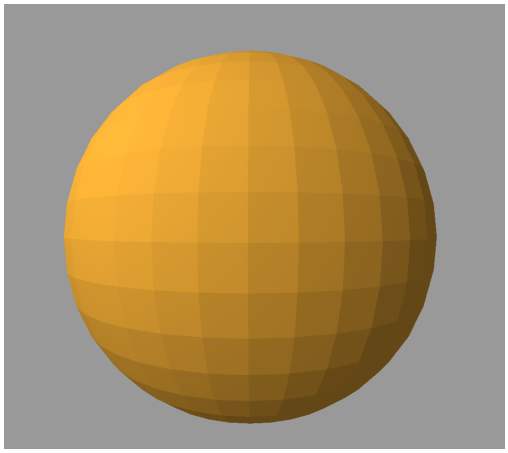
☐ **Approximately** how many fragments in the far sphere? (There is not enough information to answer this question exactly, don't try.) ☐ Answer in terms of T, V_o, F_o, and d.

Staple This Side

(*c*) The **only difference** between the two spheres below on the right is in the normal associated with each vertex. The normals for the smooth sphere were chosen to make it look more like a sphere.

☐ Complete the code below so that it inserts normals that will result in the faceted sphere.

```
// Center of sphere at sp_ctr_os;

// Coordinates of a triangle approximating
// the surface of a sphere.
//
coors_os << p1 << p2 << p3;
```

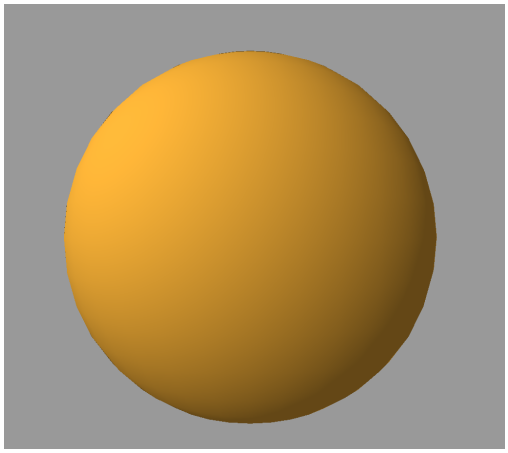


Faceted Sphere

```
normals_os <<
```

☐ Complete the code below so that it inserts normals that will result in the smooth sphere.

```
// Center of sphere at sp_ctr_os;

// Coordinates of a triangle approximating
// the surface of a sphere.
//
coors_os << p1 << p2 << p3;
```



Smooth Sphere

```
normals_os <<
```

Problem 4: [20 pts] Answer each question below.

(*a*) Answer each question about the amount of computation needed for common operations. All vectors have three components and all coordinates have four components.

☐ How many multiplies are needed to compute a dot product such as `dot(v1,v2)`?

☐ How many multiplies are needed to compute a cross product such as `cross(v1,v2)`?

☐ How many multiplies are needed to perform a coordinate transformation `m * p1` , where `m` is a transformation matrix and `p1` is a coordinate?

☐ How many multiplies are needed to compute a matrix product `m1 * m2` where `m1` and `m2` are transformation matrices?

(*b*) Given coordinate P and vector v, we can easily compute $Q = P + v$. Show a transformation matrix, $\mathbf{M}$, that achieves the same result by computing a matrix/homogeneous coordinate product, $Q = \mathbf{M}P$. (For example, with $v = (v_x, v_y, v_z) = (1, 2, 3)$ and $P = (10, 20, 30, 1)$ we would compute $Q = (11, 22, 33, 1)$).

☐ Show the transformation matrix:

☐ Describe a situation in which it would make more sense to compute $Q = P + v$ rather than $Q = \mathbf{M}P$.

☐ Describe a situation in which it would make more sense to use a translation matrix rather than just adding a vector. *Hint: we do it all the time in our code.*

Staple This Side

Staple This Side

Problem 5: [15 pts] Answer the questions below.

(*a*) The code below inserts the coordinates and colors for a cylinder:

```
for ( int i=0; i<=n_segs; i++ ) {
    // Some code omitted.
    coors_os << c + cyl_axis << c;
    colors   << color_cyan   << color_cyan;
  }
```

☐ What would the cylinder look like if the frame buffer were written with the color `color_cyan`?

☐ What should the frame buffer be written with, if not exactly `color_cyan`?

(*b*) The `Our_3D` C++ class in the course CPU-only code is modeled very loosely on a Vulkan implementation, and `World` is modeled on something like an application (such as a game) a developer would write.

☐ Who writes Vulkan and OpenGL implementations? ☐ Why do most of them give away the implementations for free?

(*c*) Rasterization and ray tracing in `Our_3D` differ in what their outer loops iterate over.

☐ What kind of an object does the outer loop of `Our_3D` rasterization code iterate over?

☐ What kind of an object does the outer loop of `Our_3D` ray tracing code iterate over?

Staple This Side