## Notes

Entire test will assume X-right, Y-up, Z-forward (away from camera). We will also assume a counter-clockwise front-face winding order.

## C++

### Bit Manipulation

Implement the following methods;

bool IsBitSet( uint flags, uint bit ); // bit is 0 to 31

bool AreAllBitsSet( uint currentFlags, uint flagsToCheck );

bool AreAnyBitsSet( uint currentFlags, uint flagsToCheck );

uint SetBit( uint flags, uint bit );

uint SetBitTo( uint flags, uint bit, bool set );

### Function & Method Pointers

* typedef the signature type that would work for this function;  
  uint OnDeathEvent( std::string const &name, EventArgs &args );
* typedef the signature for the following method;   
  bool Collider2D::IsTouching( Collider2D const \*other );
* Given the above, how would you call the method pointer on a Collider2D object?

### Templates

Implement template versions of;

**Min**, **Max**, and **Clamp** function that will work for all numeric types (int, uint, float), including your Vector2 and Vector3 types. Assume; Min and Max are component-wise for Vectors;

struct Vector2 { float x, y; }

struct Vector3 { float x, y, z; }

## Shape Factory

//------------------------------------------------------------------------

class Shape {

public:

Shape() { printf( "Shape::Shape()\n" ); }

~Shape() { printf( "Shape::~Shape()\n" ); }

virtual bool Contains( vec2 pos ) const { printf( "Shape::Contains\n" ); return false; }

};

//------------------------------------------------------------------------

class Disc : public Shape {

public:

Disc() { printf( "Disc::Disc()\n" ); }

~Disc() { printf( "Disc::~Disc()\n" ); }

virtual bool Contains( vec2 pos ) const { printf( "Disc::Contains\n" ); return false; }

vec2 m\_center;

float m\_radius;

};

//------------------------------------------------------------------------

class Box : public Shape {

public:

Box() { printf( "Box::Box()\n" ); }

~Box() { printf( "Box::~Box()\n" ); }

virtual bool Contains( vec2 pos ) const { printf( "Box::Contains\n" ); return false; }

vec2 m\_center;

vec2 m\_right;

vec2 m\_extents; // half dimensions;

};

//------------------------------------------------------------------------

enum eShapeType {

SHAPE\_TYPE\_DISC,

SHAPE\_TYPE\_BOX,

};

//------------------------------------------------------------------------

class ShapeFactory {

public:

Shape\* CreateShape( eShapeType type ) {

switch (type) {

case SHAPE\_TYPE\_DISC: return new Disc();

case SHAPE\_TYPE\_BOX: return new Box();

default: return nullptr;

}

}

void DestroyShape( Shape \*shape ) {

delete shape;

}

};

1. What is the output of the following code?

void Foo( vec2 pos ) {

Shape \*shape = g\_theShapeFactory->CreateShape( SHAPE\_TYPE\_BOX );

if (shape->Contains( pos ) {

printf( "Hi!\n" );

}

g\_theShapeFactory->DestroyShape( shape );

}

* Implement Disc::Contains
* Implement Box::Contains
* Implement the following constructor for Box;  
  Box::Box( float center, vec2 fullSize, float rotationDegrees );
* Implement: bool IsTouching( Disc const &disc0, Disc const &disc1 );
* Implement: bool IsTouching( Disc const &disc, Box const &box );
* Implement: bool IsTouching( Box const &box0, Box const &box1 );
* Implement: uint Disc::Raycast( float \*outResult, ray2 ray ) const;
* Implement: uint Box::Raycast( float \*outResults, ray2 ray ) const;

## Math (Misc)

* Implement: Vector2 Camera::WorldToScreenPoint( vec3 pos );
* Implement: Ray3 Camera::ScreenToWorldRay( vec2 screenPos );
* Given three points, generate a plane (3D). Define Plane3 and the following function;   
  // a, b, c form a triangle drawn counter-clockwise;

Plane3 FromPoints( Vector3 a, Vector3 b, Vector3 c );

* Implement raycast against a plane (3D)  
  uint Raycast( float \*outResult, Ray3 ray );
* Implement: Ray3 Reflect( Ray3 ray, Plane3 plane );  
  Should return a ray starting at the collision point, with the reflected direction; If no collision, return the same ray;

## Word Problems: Dot Products and Cross Products

Assume transform2 which has a position and a rotation (degrees). X is right, Y is up.

* Implement the following function   
  float GetDamageMultiplier( transform2 const &attacker, transform2 const &victim );
  + Design wants the rogue to do 2x damage from the back
  + Design wants the rogue to do 4x damage in a 30 degree cone behind the target, 1x for a 90 degree cone in front, and 2x if attacked on the sides;
* For a platformer, design sets a slope (rise/run) of a max walkable slope before the character starts “falling” or slipping. Assume world is made up of clock-wise polygons, and you have a raycast method that will return the polygon and line. You are currently in the “standing” state, determine if you should move to the “falling” state by calling SetMoveState( MOVE\_STATE\_FALLING )  
  void Entity::UpdateStanding() { … }
  + Assume the world can rotate arbitrarily (have an arbitrary down and gravity). Update the above to account for that.
* We have a 2D tank game where we can fire homing rockets. The rockets have a targeting cone (coneDegrees) in which any target in that cone they will home in on. They are also given a turn speed (degreesPerSecond) and speed (unitsPerSeconds) to move.
  + Entity\* Rocket::FindTarget( std::vector<Entity\*> const &entities ) const;   
    Return any target in the cone.
  + void Rocket::Update( float deltaTimeSeconds )  
    If no target- find one. If you have a target, turn toward it. Also update position based on speed;

Assume transform3, containing a mat44 representing its world transform (position and rotation).

* 3D Space Dogfighter game. Your ship is represented by two radii; a shield radius, and a hull radius. You ship has shields in front, back, left, and right, each with its own energy value. The hull is only damaged if the shield has no energy on that side;   
    
  Both Ship and Projectile have a mat44 m\_worldTransform;   
    
  Implement   
  bool Ship::CheckContact( Projectile \*proj );   
    
  Which if it damages the shield, call DamageShield( eShieldSide ); If it damages the hull call DamageHull();
  + Implement a hit indicator to show the player where the shot is coming from. If the shot is in front from directly in front (front 10 degree cone), do nothing but flash the screen (FlashScreen()). If the shot is from the front/side, so a small arrow in the direction the player should turn. If from behind, show a large arrow in the direction to turn. Direction to turn should be the direction of shortest distance.

## Rendering State

The stages of the pipeline we dealt in this class were;  
Input Assembler : IA  
Vertex Shader: VS  
Rasterizer: RS  
Pixel (Fragment) Shader: PS/FS  
Output-Merger (Framebuffer): OM/FB

Which stage is the following state/data bound to (may have more than one answer);

* Shader Resources (Textures/Constant Buffers)
* Input Layout
* Blend State
* Render Targets
* Cull Winding Order
* Depth-Stencil State
* Vertex Buffers
* Fill Mode
* Index Buffers
* ???

### Blend Modes

We’ll deal only with color blending; Blending is made up of an operation, and a source and destination factor. Whare is the op, src, and dst for the following named blending modes;

* Opaque
* Alpha
* Additive

### Depth/Stencil State

With depth, we cared about three things. Is depth testing enabled, what is the test, and should we write our result; Give pseudo code for how the depth state is working;

static const bool DEPTH\_ENABLED;  
static const bool DEPTH\_WRITE;   
static bool (\*DepthOp)( float srcDepth, float dstDepth );

Image gColorTarget;   
Image gDepthTarget;

void DoDepthWork( ivec2 texelCoord, float4 outputColor, float outputDepth )  
{  
 // …

}

* Depth testing is not usually used for objects rendered with Alpha. Why?
* How should alpha objects be rendered?
  + How does this change if we are rendering additively?

## Meshes

Generate positions, UVs, normal, and tangents.

* What is the purpose of the input layout?
* Generate a mesh for Cube
* Generate a mesh for a UV Sphere
* Generate a mesh for a torus (donut)

## Matrices

* Give a matrix for rotation around Z (k);
* Give a matrix for rotation around Y (j);
* How do you generate a look-at matrix (given a position, a target, and a world up)
* What is the job of the projection matrix?
* Assume you have a camera containing the view and projection matrices, as well as an output texture…
  + Draw a billboard quad facing the camera (give the vertices and indices)

## Shaders

* Mesh is passing up positions, normal, and tangent in local (model) space. Write the vertex shader such that you forward the SV\_POSITION (clip space), a WORLD\_POSITION, WORLD\_NORMAL, and WORLD\_TANGENT;
* Pixel Shader: Given a WORLD\_NORMAL and WORLD\_TANGENT, calculate the normal to use, using a normal map;
* Assuming you have an infinite point light (no falloff), calculate diffuse light.
  + Calculate specular light
* How are the light values used to determine the final color?
* Give code for outputting a UV as a color
* Give code for outputting a world vector as a color;
* If you were doing some complex math inside a shader, and you want to check the result after a certain step – how would you go about this? (assume the result is a single float).