## 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)

Assume you have a Camera that contains an output target (Texture2D), and a view and projection matrix (Matrix4x4).

* Implement: Vector2 Camera::WorldToScreenPoint( vec3 pos );
* Implement: Ray3 Camera::ScreenToWorldRay( vec2 screenPos );

Just general math…

* 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.

### Back Stabs

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;

### Platformer Slope

For a platformer, design sets a slope (rise/run) of a max walkable slope before the character starts “falling” or slipping.

You are currently in the *standing* state. Determine if you should move to the *falling* state by calling SetMoveState( MOVE\_STATE\_FALLING ).

You may assume you have the following helper functions;

float const WALKABLE\_SLOPE; // 0 means completely flat, defnied as "rise/run",

struct segment2 { vec2 start; vec2 end; };

struct polygon2 { std::vector<Vector2> points; }

bool World::Raycast( polygon2 \*outPoly, segment2 \*outSegment, ray2 worldRay );

Implement the following;

void Entity::UpdateStanding() {

// implement me - call SetMoveState( MOVE\_STATE\_FALLING ) when

// not on a walkable surface.

// ...

}

### Tank Missiles

We have a 2D tank game where we can fire homing rockets. The rockets have a targeting cone (m\_coneDegrees) in which any target in that cone they will home in on. They are also given a turn speed (m\_degreesPerSecond) and speed (m\_unitsPerSeconds) to move.

Entity provides the following helpers;

vec2 Entity::GetPosition() const;

vec2 Entity::GetForward() const;

float Entity::GetRotation() const;

void Entity::MoveBy( vec2 offset );

void Entity::RotateBy( float degrees );

Rock holds onto its current target (Entity\* Rocket::m\_currentTarget;)

Implement the following methods;

* Entity\* Rocket::FindTarget( std::vector<Entity\*> const &entities ) const;
  + *Note: In an interview setting, a good topic of discussion is how to best choose a target in the cone; For now, any target in the cone is fine;*
* void Rocket::Update( float deltaTimeSeconds )
  + If target is outside the cone – lose the target.
  + Should find a new target if there is no current target;
  + If target exists, turn toward the target
  + Move along current heading

### 3D Shields

Given a **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;

Given…

enum eShieldSide {

SHIELD\_FRONT,

SHIELD\_RIGHT,

SHIELD\_BACK,

SHIELD\_LEFT,

};

class Projectile {

public:

Matrix4x4 m\_transform;

vec3 m\_previousPosition;

};

class Ship {

public:

float GetShieldEnergy( eShieldSide side ) const;

float DamageShield( eShieldSide side );

float DamageHull();

bool CheckContact( Projectile \*proj );

private:

Matrix4x4 m\_transform;

float m\_hullRadius;

float m\_shieldRadius;

};

1. Implement the following;

bool CheckContact( Projectile \*proj ) {

// assume shield is a shell.

// if shield is hit, call DamageShield for the side that was damaged

// if shield is down (no energy), damage hull if hull is hit (DamageHull())

// objects are allow to pass safely out of a shield

// ...

}

1. Implement a hit indicator to show the player where the shot is coming from. If the shot is hitting the player from the front 10 degree cone do nothing but flash the screen (FlashScreen()). If the shot is from the front/side, draw 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.   
     
   You are given the following helpers;  
   void Game::FlashScreen();   
     
   // short is .5f, long is 1.0f. 0 degrees makes an arrow pointing

// to the right, and rotates counter-clockwise;

void Game::FlashArrow( float length, float degrees );

## 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
* Samplers

### Blend Modes

We’ll deal only with color blending; Blending is made up of an operation, and a source and destination factor. The following all use the *add* operation, what are the source and destination factors?

Options are: ZERO, ONE, SRC\_COLOR, SRC\_ALPHA, ONE\_MINUS\_SRC\_ALPHA,  
 DST\_COLOR, DST\_ALPHA, ONE\_MINUS\_DST\_ALPHA

* 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 step works;

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 )

{

// ...

}

## 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

## 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

### Debugging

* Give code for outputting a UV as a color;   
  float4 UVAsColor( float2 uv ) { /\*...\*/ }
* Give code for outputting a world vector as a color;   
  float4 WorldUnitVectorAsColor( float3 unitVector ) { /\*...\*/ }
* 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).

### Lighting

You are given a shader with the following input;

struct vs\_input\_t

{

float3 position : POSITION;

float3 normal : NORMAL;

float4 tangent : TANGENT;

float2 uv : UV;

};

Write a vertex shader that outputs the following…

struct v2f\_t

{

float4 position : SV\_Position;

float3 normal : NORMAL;

float3 tangent : TANGENT;

float3 bitangent : BITANGENT;

float2 uv : UV;

float3 world\_position : WORLD\_POSITION;

};

…given the following constants and resources;

Texture2D <float4> tDiffuse : register(t0);

Texture2D <float4> tNormal : register(t1);

Texture2D <float4> tSpecular : register(t2);

Texture2D <float4> tEmissive : register(t3);

SamplerState sLinear : register(s0);  
  
cbuffer matrix\_constants : register(b2)  
{  
 float4x4 MODEL;  
 float4x4 VIEW;  
 float4x4 PROJECTION;  
};

Implement:

v2f\_t VertexFunction(vs\_input\_t input)

{

v2f\_t v2f = (v2f\_t)0;

// ...

return v2f;

}

Implement calculating a world normal from a normal map give the previous fragment shader input;

float4 FragmentFunction( v2f\_t input ) : SV\_Target0

{

float4 normalColor = tNormal.Sample( sLinear, input.uv );

float3 worldNormal = float3(0, 0, 1);

// calculate world normal

// ...

return WorldUnitVectorAsColor( worldNormal );

}

Assuming you have an infinite point light (no falloff), calculate diffuse light. Define the constant buffer you would use, as well as implement the lighting using the worldNormal lighting;

1. Diffuse factor
2. Specular Factor
3. Final Color

float4 FragmentFunction( v2f\_t input ) : SV\_Target0

{

float4 normalColor = tNormal.Sample( sLinear, input.uv );

float3 worldNormal = float3(0, 0, 1);

// calculate world normal

float3 diffuseLighting = float3(0, 0, 0);

// ...

float3 specularLighting = float3(0, 0, 0);

// ...

float4 finalColor = /\*...\*/;

return finalColor;

}