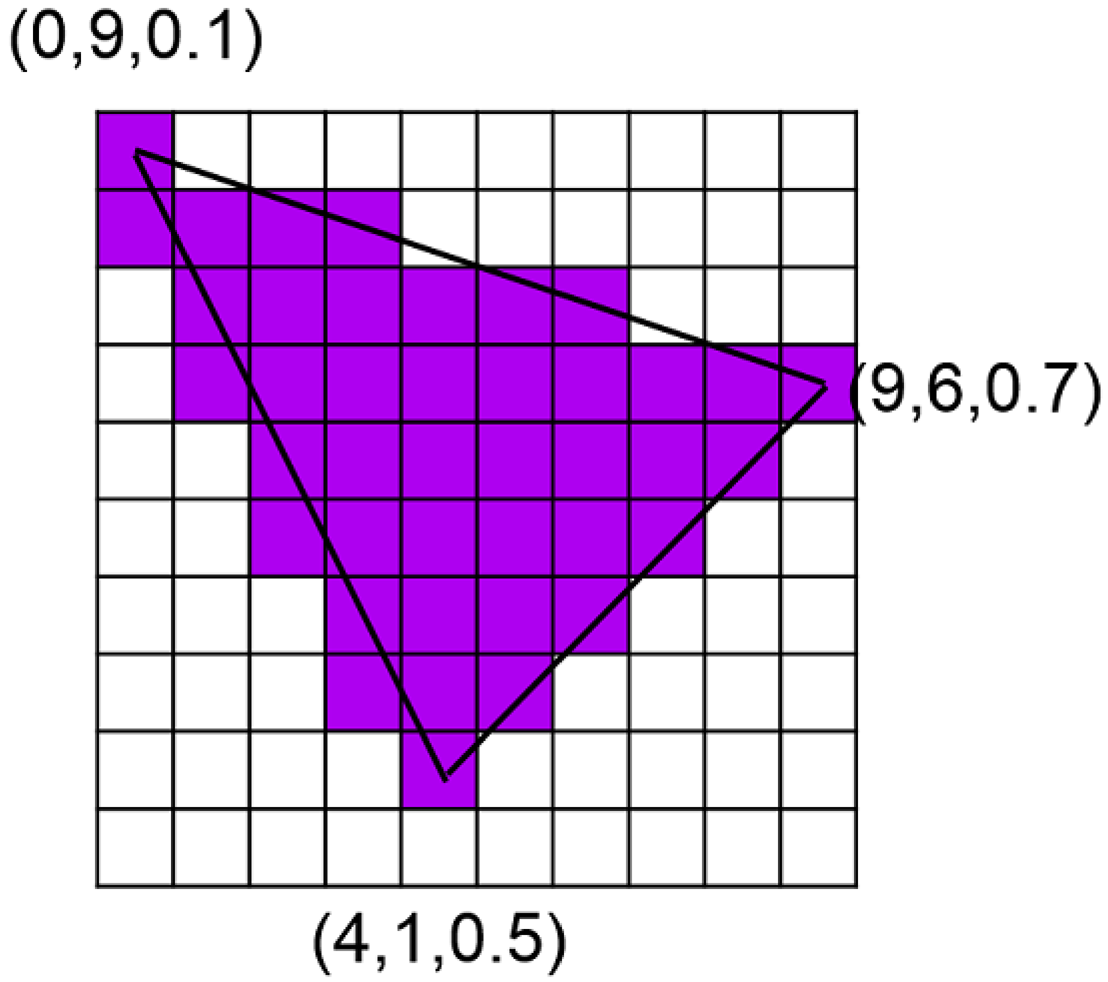
Week 5 Tutorial Solutions

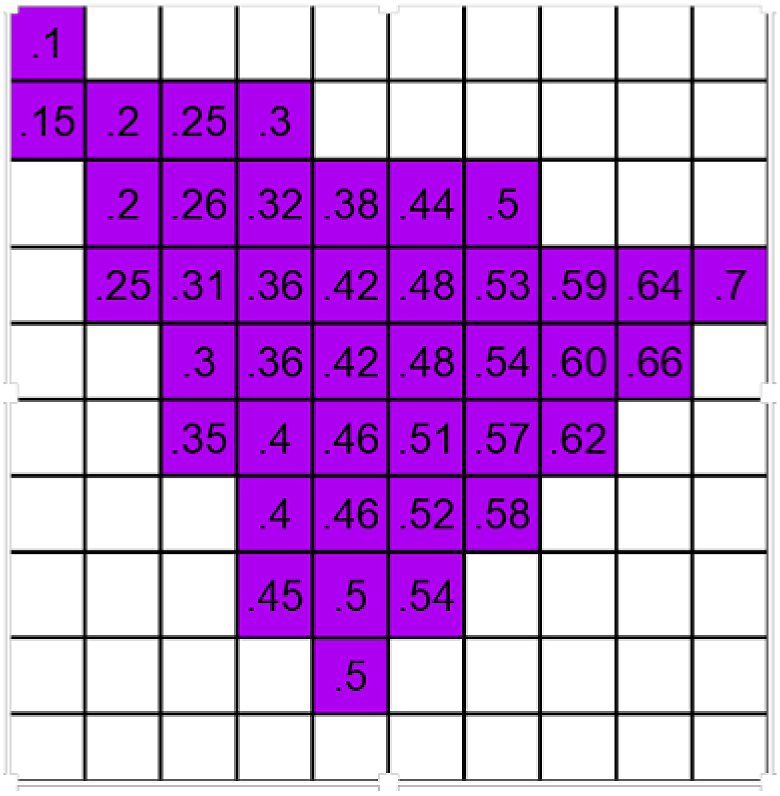
Question 1: Depth buffer

Use bilinear interpolation to work out the pixel depths for the triangle below, given the coordinates and pseudo-depth values for the vertices as stated. Calculate depth values to two decimal places.

NOTE: Feel free to only calculate the depth of 1 or 2 pixels if that's all you need to understand the process.

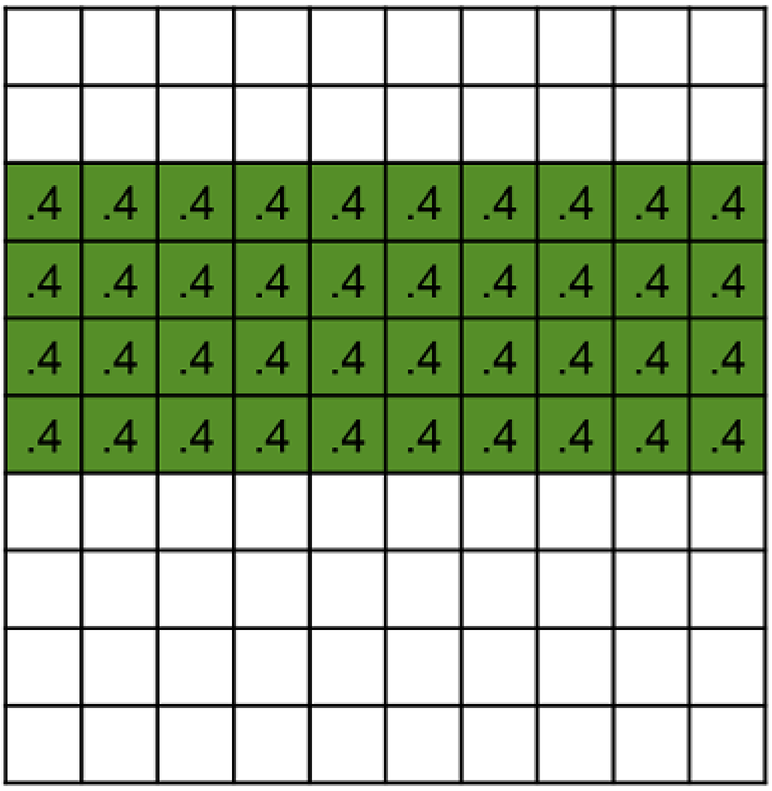


Interpolated depths:



.1									
.15	.2	.25	.3						
	.2	.26	.32	.38	.44	.5			
	.25	.31	.36	.42	.48	.53	.59	.64	.7
		.3	.36	.42	.48	.54	.60	.66	
		.35	.4	.46	.51	.57	.62		
			.4	.46	.52	.58			
			.45	.5	.54				
				.5					

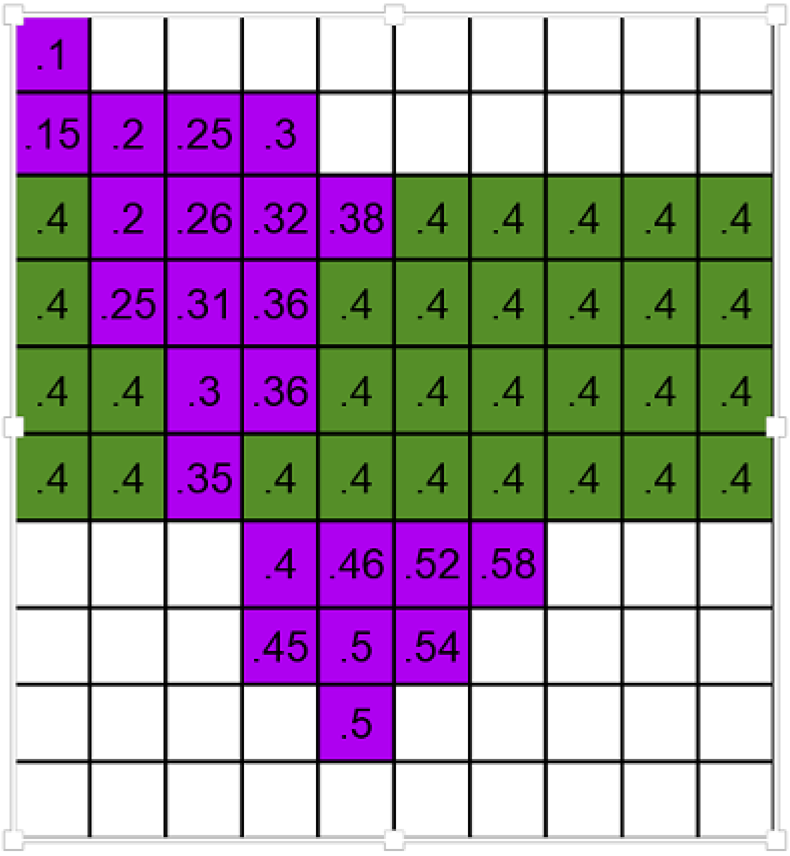
Show the result of copying the triangle into the depth buffer that already has the rectangle below in it



.4	.4	.4	.4	.4	.4	.4	.4	.4	.4
.4	.4	.4	.4	.4	.4	.4	.4	.4	.4
.4	.4	.4	.4	.4	.4	.4	.4	.4	.4
.4	.4	.4	.4	.4	.4	.4	.4	.4	.4

Image drawn with depth buffer:

Note that by default depth buffer values range from 0 for near to 1 for far. And also by default each fragment must have a depth LESS_THAN the pseudodepth in the buffer otherwise it is not drawn.



.1									
.15	.2	.25	.3						
.4	.2	.26	.32	.38	.4	.4	.4	.4	.4
.4	.25	.31	.36	.4	.4	.4	.4	.4	.4
.4	.4	.3	.36	.4	.4	.4	.4	.4	.4
.4	.4	.35	.4	.4	.4	.4	.4	.4	.4
			.4	.46	.52	.58			
			.45	.5	.54				
				.5					

Question 2: Illumination

Suppose you have modelled a vertex with the following material properties.

```
ambientCoeff = 0.1
diffuseCoeff = 0.5
specularCoeff = 1
phongExp = 10
```

And a light source with the following properties

```
lightPos = (0,3,2)
lightIntensity = 1
ambientIntensity = 0.2
```

Assume

- The vertex is at position(0,0,-2)
- The normal at the vertex is (0,0,1)

Assume these have already been transformed into camera/eye co-ordinates.

a. Calculate the diffuse component for the vertex

   For this we need the direction to the lightsource(s) and the normal (m). These should both be normalised. We then need to take their dot product and multiply by the diffuse co-efficients for the material and the lighting.

   ```
   s = (0,3,2) - (0,0,-2) = (0,3,4)
   |s| = sqrt(0+9+16) = 5
   s_unit = (0,0.6,0.8)

   s_unit.m = (0,0.6,0.8) . (0,0,1) = 0.8

   diffuse = 0.5*1*0.8 = 0.4
   ```

b. Calculate the specular component assuming we are using the phong specular model.

For this we need the reflected vector which we need the direction to the lightsource(s) and the direction to the view/camera (v). We already calculated the direction to the light source. Since we are in camera co-ordinates the view/camera is at (0,0,0)

```
v = (0,0,0) - (0,0,-2) = (0,0,2)
|v| = sqrt(4) = 2
v_unit = (0,0,1)
r = -s + 2*(s . m)m = -(0,3,4) + 2((0,3,4) . (0,0,1))(0,0,1) = (0,-3,4)
r_unit = (0,-0.6,0.8)

r_unit.v_unit = (0,-0.6,0.8).(0,0,1) = 0.8
pow(0.8,10) = 0.11
specular = 1*1*0.11 = 0.11
```

c. Calculate the total intensity for the vertex.

```
ambient = 0.2*0.1 = 0.02
intensity = ambient + diffuse + specular
          = 0.02 + 0.4 + 0.11
          = 0.53
```

Questionn 3: Blinn-Phong

Recall the gouraud vertex shader from lectures:

```
// Incoming vertex position
in vec3 position;

// Incoming normal
in vec3 normal;

uniform mat4 model_matrix;

uniform mat4 view_matrix;

uniform mat4 proj_matrix;

// Light properties
uniform vec3 lightPos;
uniform float lightIntensity;
uniform float ambientIntensity;

// Material properties
uniform float ambientCoeff;
uniform float diffuseCoeff;
uniform float specularCoeff;
uniform float phongExp;

out float intensity;

void main() {
    // The global position is in homogenous coordinates
    vec4 globalPosition = model_matrix * vec4(position, 1);

    // The position in camera coordinates
    vec4 viewPosition = view_matrix * globalPosition;

    // The position in CVV coordinates
    gl_Position = proj_matrix * viewPosition;

    // Compute the normal in view coordinates
    vec3 m = normalize(view_matrix*model_matrix * vec4(normal, 0)).xyz;

    // Compute the s, v and r vectors
    vec3 s = normalize(view_matrix*vec4(lightPos,1) - viewPosition).xyz;
    vec3 v = normalize(-viewPosition.xyz);
    vec3 r = normalize(reflect(-s,m));
    float ambient = ambientIntensity*ambientCoeff;
    float diffuse = max(lightIntensity*diffuseCoeff*dot(m,s), 0.0);
    float specular;

    // Only show specular reflections for the front face
    if (dot(m,s) > 0)
        specular = max(lightIntensity*specularCoeff*pow(dot(r,v),phongExp), 0.0);
    else
        specular = 0;

    intensity = ambient + diffuse + specular;
}
```

- Make sure you understand how the shader is applying the lighting calculations.
- Change the shader so it using the Blinn-Phong model of specular reflection instead of the Phong model. In Blinn-Phong, we use the halfway vector **h** instead of the reflected vector **r**. You can calculate **h** by averaging the **s** and **v** vectors.

  To do this, we simply need to change this line

  ```
  vec3 r = normalize(reflect(-s,m));
  ```

  to

  ```
  vec3 h = (s + v)/2;
  ```

  and

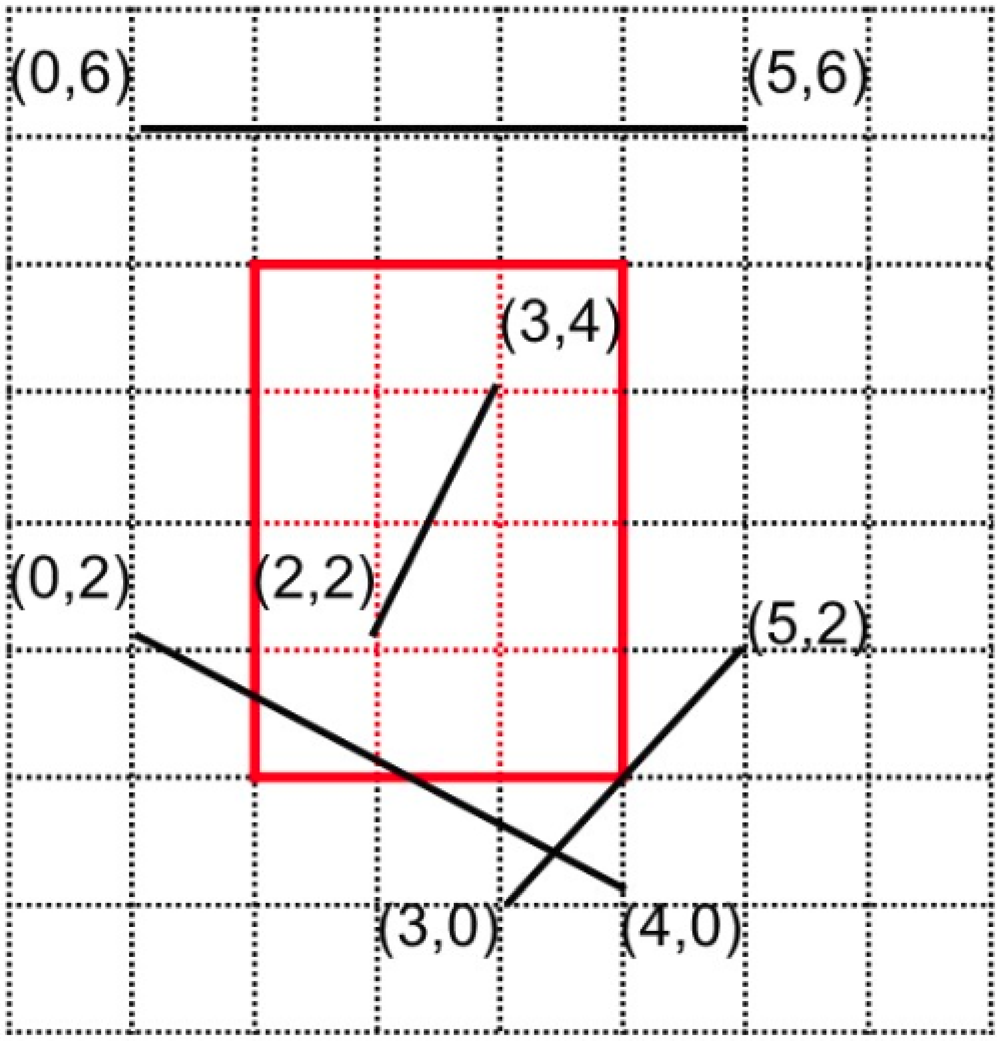
  ```
  specular = max(lightIntensity*specularCoeff*pow(dot(r,v),phongExp), 0.0);
  ```

  to

  ```
  specular = max(lightIntensity*specularCoeff*pow(dot(h,m),phongExp), 0.0);
  ```

Question 4: Clipping

Given the clipping rectangle with bottom-left corner (1,1) and top-right corner (4,5), apply the Cohen-Sutherland algorithm to clip the following lines:



Compute endpoint codes in the order (Top, Bottom, Left, Right) with outside = 1, inside = 0

A = (0,2), B=(4,0)

End(A) = (0,0,1,0)
End(B) = (0,1,0,0)

There is no bit in common, so we need to clip.
Clip A with left -> A = (1, 1.5)

End(A) = (0,0,0,0)
End(B) = (0,1,0,0)

We need to clip again.
Clip B with bottom -> B = (2,1)

End(A) = (0,0,0,0)
End(B) = (0,0,0,0)

This is now a trivial accept.

A = (3,4), B=(2,2)

End(A) = (0,0,0,0)
End(B) = (0,0,0,0)

Both are zero ->Trivial accept. No clipping needs to be done.

A = (0,6), B=(5,6)

End(A) = (1,0,1,0)
End(B) = (1,0,0,1)

The top bit is 1 in both -> Trivial reject

A = (3,0), B=(5,2)

End(A) = (0,1,0,0)
End(B) = (0,0,0,1)

No bit in common -> we need to clip.

Clip A to bottom -> A = (4, 1)

End(A) = (0,0,0,0)
End(B) = (0,0,0,1)

No bit in common -> we need to clip

Clip B to right hand side -> B = (4,1)

End(A) = (0,0,0,0)
End(B) = (0,0,0,0)

Both are zero -> Trivial accept. No clipping needs to be done.

Repeat the process using the Cyrus-Beck algorithm instead.

A = (0,2), B=(4,0)

```
(B-A) = (4, -2)

tmin = 0
tmax = 1

1) Intersect with left edge, P = (1,3), n = (-1, 0)

t = n . (P-A) / n . (B-A)
  = 0.25

n . (B-A) = (-1, 0) . (4, -2)
          = -4
          < 0  so the ray it entering

tmin = max(0, 0.25) = 0.25
tmax = 1

2) Intersect with top edge, P = (2,5), n = (0,1)
t = -1.5
n . (B-A) = -2 < 0 so the ray is entering

tmin = max(0.25, -1.5) = 0.25
tmax = 1

3) Intersect with right edge, P = (4,3), n = (1,0)
t = 1
n . (B-A) = 4 > 0 so the ray is exiting

tmin = 0.25
tmax = min(1,1) = 1

4) Intersect with bottom edge, P = (2, 1), n = (0, -1)
t = 0.5
```

```
n . (B-A) = 2 > 0 so the ray is exiting

tmin = 0.25
tmax = min(1, 0.5) = 0.5

Done. Edge is clipped to:

L(0.25) = (1, 1.5)
L(0.5) =  (2, 1)
```

The rest are left as an exercise to the reader.

A = (3,4), B=(2,2)

A = (0,6), B=(5,6)

A = (3,0), B=(5,2)