

## **Midterm**

**You have 1 hour, 15 minutes to complete this exam**

This exam is closed book and closed notes. The number in brackets (e.g. [5]) tells you how many points each problem is worth. If there are several parts, I sometimes break it down for each part to give you an idea of how I will grade it. **This exam might be long for some of you.** The pages are double-sided and there are 45 problems in total, which gives you a 1:40 to work on each problem. Some of the problems you will answer quickly, which will save you time for the problems that take longer.

**The exam is worth a total of 186 points.** I suggest that in order to maximize your score, you work fast and efficiently. It would help to go through and answer all the questions you can answer quickly first, and then go back to the questions you have trouble with later. If you need more space, use the blank sheets of paper provided and label your work with the problem number (and indicate that your work is continued on the attached sheet next to the problem here). Do your best, I will normalize the grades at the end.

SOLUTIONS

---

Name

1. [1] Write the matrix  $M$  that will translate the current coordinate system so that the new origin is now at point  $(a,b,c)$ .

$$M = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. [1] Given the matrix  $M$  from above, where would point  $(d, e, f)$  be transformed to?

$$(d+a, e+b, c+f)$$

3. [1] Compute the result of multiplying the given vector by the following matrix.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$$

4. [1] What is the result of the following matrix-matrix multiplication?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

5. [1] If  $v_1$  is vector  $[a, b, c]^T$  and  $v_2$  is vector  $[d, e, f]^T$  what is the dot product  $v_1^T \cdot v_2$ ?

$$ad+be+cf$$

6. [4] Describe the following coordinate spaces as defined by OpenGL:

[2] Eye-space coordinates

Camera is placed at the origin looking down at the  $-z$  axis.  $+x$  is to the right,  $+y$  is up.

[2] Screen-space coordinates

Origin is at lower left corner of the screen.  $+x$  is to the right,  $+y$  is up. Vertices now range from 0 to screen width (for  $x$ ) and 0 to screen height (for  $y$ ).

7. [5] What are homogeneous coordinates [2]? Why do we use them in computer graphics [3]?

Adding a fourth component " $w$ " to form coordinates of the type  $(x, y, z, w)$ . In homogeneous coordinates, we define  $(x, y, z, w) = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$ .

This represents point  $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$  in 3-D space. We use homogeneous coordinates in graphics so that we can represent affine transformations  $ax+b$  (such as translation) by a matrix. Without it, we could not translate points by multiplying by a matrix.

8. [7] If you transform a vertices of a model by modelview matrix  $M$ , what matrix should you multiply its normals by to get a correct result [2]? Remember, the normals should remain perpendicular to the surface after the transformation. How do you derive this, as we showed in class [5]?

You should multiply the normals by the inverse transpose of the modelview Matrix ( $M^{-T}$ )

$$n^T v = 0 \leftarrow \text{normals should be } \perp \text{ to surface}$$

$$(M_n n)^T M v = 0 \leftarrow \text{after transformation they should remain } \perp$$

$$n^T \underbrace{M_n^T M}_I v = 0$$

$$M_n^T M = I$$

$$M_n^T M \cdot M^{-1} = M^{-1}$$

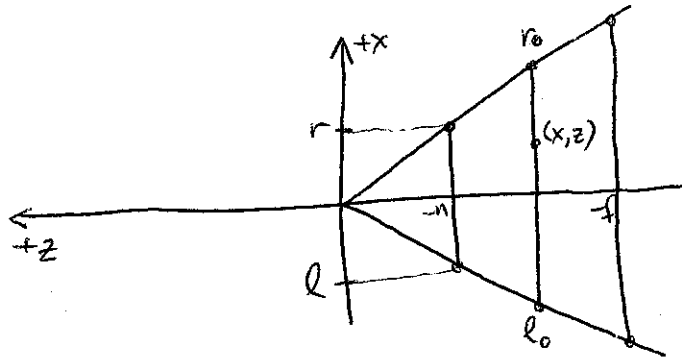
$$M_n^T = M^{-1}$$

$$\boxed{M_n = (M^{-1})^T}$$

9. [3] Define the term "rendering."

Rendering  $\Rightarrow$  the process of taking a description of the scene (it could be geometry, shaders, camera models, even images) and generating an image of that scene from the specified view.

10. [10] Derive the perspective projection transformation functions for the  $x$  and  $y$  terms only. Assume that (as in OpenGL), the frustum is defined in  $x$  by the left plane  $l$  and the right plane  $r$  and in the  $y$  axis by the bottom plane  $b$  and the top plane  $t$ . Unlike OpenGL, set up the transformation so that it maps the points to the canonical volume that ranges from 0 to 1. Write the functions in the form  $x' = f(x,y,z)$  and  $y' = f(x,y,z)$ , where  $x'$  and  $y'$  are the transformed  $x$  and  $y$  coordinates.



by similar triangles:

$$\frac{r_0}{z} = \frac{r}{-n} \Rightarrow r_0 = -\frac{rz}{n}$$

$$\frac{l_0}{z} = \frac{l}{-n} \Rightarrow l_0 = -\frac{lz}{n}$$

$$x' = \frac{1}{r_0 - l_0} (x - l_0) \leftarrow \text{maps } l_0 \leq x \leq r_0 \text{ to } 0 \leq x' \leq 1$$

plug in values for  $r_0$  and  $l_0$

$$x' = \frac{n}{-rz + lz} \left( x + \frac{lz}{n} \right)$$

$$x' = \left( \frac{n}{l-r} \right) \frac{x}{z} + \frac{l}{l-r}$$

$$y' = \left( \frac{n}{b-t} \right) \frac{y}{z} + \frac{b}{b-t}$$

11. [10] Derive the general orthographic projection matrix that projects the rectangular block  $(l, r), (b, t), (-n, -f)$  in eye space into the rectangular block  $(x_0, x_1), (y_0, y_1), (z_0, z_1)$ .

Do  $x$  first:  $x' = \frac{1}{r-l}(x-l) \leftarrow \text{maps to } 0 \leq x' \leq 1$

scale by  $x_1 - x_0$ , add  $x_0$

$$x' = \frac{x_1 - x_0}{r-l}(x-l) + x_0 \leftarrow \text{maps to } x_0 \leq x' \leq x_1$$

rewrite:  $x' = \frac{x_1 - x_0}{r-l}x - \frac{l(x_1 - x_0)}{r-l} + x_0$

$y$  is the same  $\rightarrow y' = \frac{y_1 - y_0}{t-b}y - \frac{b(y_1 - y_0)}{t-b} + y_0$

$z$  is a little trickier because of negative signs on  $n$  and  $f$

$$z' = \frac{1}{f-n}(-z-n) \leftarrow \text{maps to } 0 \leq z' \leq 1$$

$$z' = \frac{z_1 - z_0}{f-n}(-z-n) + z_0 = \frac{z_1 - z_0}{n-f}z - \frac{n(z_1 - z_0)}{f-n} + z_0$$

$$M_0 = \begin{bmatrix} \frac{x_1 - x_0}{r-l} & 0 & 0 & -\frac{l(x_1 - x_0)}{r-l} + x_0 \\ 0 & \frac{y_1 - y_0}{t-b} & 0 & -\frac{b(y_1 - y_0)}{t-b} + y_0 \\ 0 & 0 & \frac{z_1 - z_0}{n-f} & -\frac{n(z_1 - z_0)}{f-n} + z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

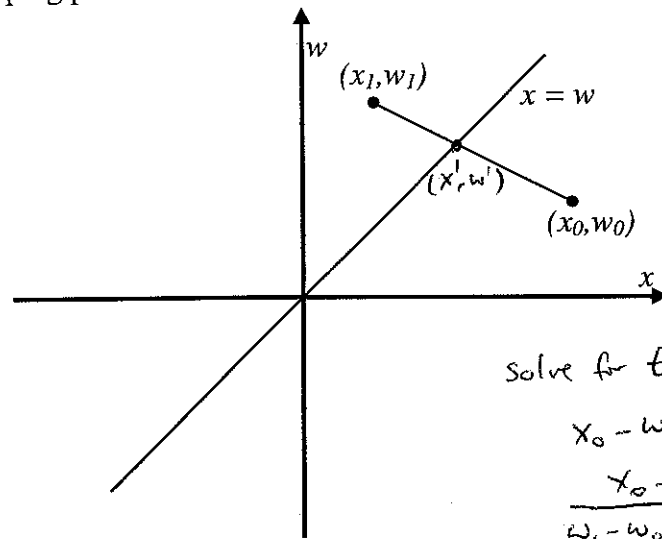
12. [5] Using the matrix you derived in the previous problem, plug in  $(-1, 1), (-1, 1), (-1, 1)$  for  $(x_0, x_1), (y_0, y_1), (z_0, z_1)$  and show that you get OpenGL's orthographic projection matrix.

plug in  $-1, 1 \rightarrow$

$$M_0 = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{2l}{r-l} + x_0 \\ 0 & \frac{2}{t-b} & 0 & -\frac{2b}{t-b} + y_0 \\ 0 & 0 & \frac{2}{n-f} & -\frac{2n}{f-n} + z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is the same as OpenGL's ortho matrix!

13. [5] During clipping, we clip line segments to the planes corresponding to the different sides of the frustum. Compute the new vertex when you clip the following line segment to the  $x = w$  clipping plane.



write parametric equations:

$$\begin{aligned} X &= X_0 + (X_1 - X_0)t \\ w &= w_0 + (w_1 - w_0)t \end{aligned}$$

set  $x = w$

$$X_0 + (X_1 - X_0)t = w_0 + (w_1 - w_0)t$$

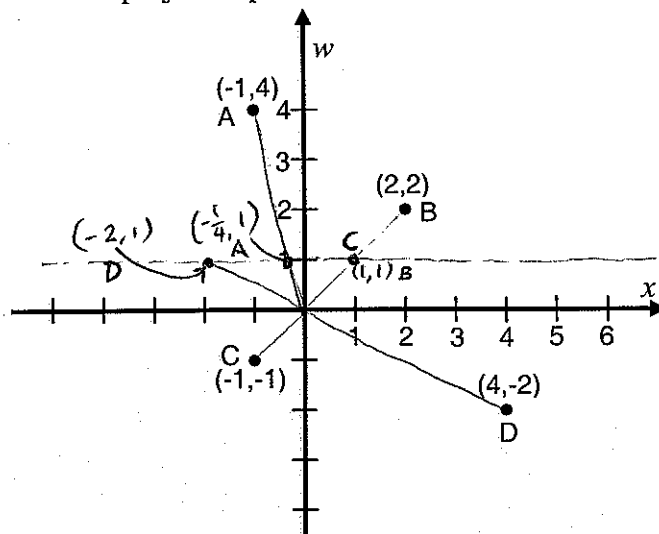
solve for  $t$ :

$$X_0 - w_0 = (w_1 - w_0 - X_1 + X_0)t$$

$$\frac{X_0 - w_0}{w_1 - w_0 - X_1 + X_0} = t$$

$$\begin{aligned} X' &= X_0 + \frac{(X_1 - X_0)(X_0 - w_0)}{w_1 - w_0 - X_1 + X_0} \\ w' &= w_0 + \frac{(w_1 - w_0)(X_0 - w_0)}{w_1 - w_0 - X_1 + X_0} \end{aligned}$$

14. [5] In the following diagram, show where the four points would project to in the  $x$ - $w$  space after performing the homogenous divide. Write down the coordinates of the final coordinates next to each projected point.



15. [4] Name four different values that are commonly interpolated by the rasterizer for use at each fragment.

color, depth, normals, texture coordinates

Write out the value stored in register *res* after the following fragment programs are executed independently of each other. Assume all the registers have been declared and initialized in the following manner:  $R1 = \{1, 2, 3, 4\}$ ,  $R2 = \{0, -1, 2, -2\}$ ,  $R3 = \{1, 1, 1, 1\}$ ,  $res = \{10, 11, 12, 13\}$ . If the instruction is a syntax error, indicate so.

16. [2] ADD *res*, R1, R2;

$res = \{1, 1, 5, 2\}$

17. [2] ADD *res*, R1.x, R2.y;

$res = \{0, 0, 0, 0\}$

18. [2] MUL *res.z*, R1, R2.y;

$res = \{10, 11, -3, 13\}$

19. [2] MUL *res.xz*, R1, R2.y;

$res = \{-1, 11, -3, 13\}$

20. [2] SUB *res.z*, R1, R2.xy;

SYNTAX ERROR! INVALID SWIZZLE

21. [2] CMP *res*, R2.y, R2.x, R2.z;

$res = \{0, 0, 0, 0\}$

22. [2] DP3 *res*, R1, R3;

$res = \{6, 6, 6, 6\}$

23. [2] DP4 *res*, R1, R3;

$res = \{10, 10, 10, 10\}$

For the following questions, you are executing a fragment program on a full-screen-sized quad and the texture coordinates  $uv$  range from 0 to 1, with (0,0) in the bottom-left corner of the rendered quadrangle. The  $z$  and  $w$  components of  $uv$  are 0 and 1, respectively. What are the outputs of the following fragment programs? In other words, what do the shaded surfaces look like? Describe it with words or draw a picture if it helps to describe it. Try to be as precise as possible (e.g. label the dimension of things if they are known, and identify the output color in {r, g, b, a} format if it is known). The header of each fragment program is always the same:

```
!!ARBfp1.0
ATTRIB uv                               = fragment.texcoord[0];
# so uv is {u, v, 0, 1}
```

The different fragment programs are given below.

```
24. [1]
MOV result.color, {1, 1, 0, 0};
END
```

The output will be a full-screen YELLOW (1,1,0) quad.

```
25. [2]
MOV result.color, uv.x;
END
```

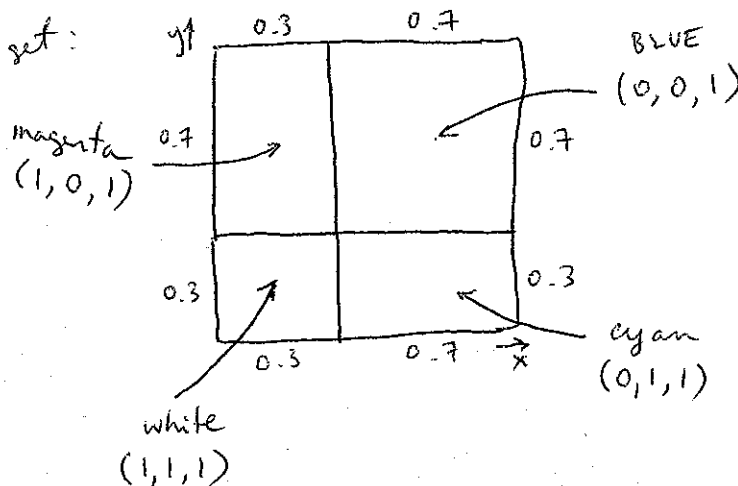
You will get a gradient going from left to right.  
It will be black on the left and linearly increase until it gets to white on the right:



```
26. [2]
TEMP t;
SUB t, uv, 0.3;
CMP result.color, t, 1, 0;
END
```

Whenever the  $uv$  elements are less than 0.3, you will get 1, else you get 0.

So you will get:



COMPUTE LINE EQUATION  
BETWEEN (0.5, 1) and (0, 0.5)

```
27. [5]
TEMP v0, v1, t, uv_temp;
```

```
{
MOV v0, {0.5, 1, 1, 0};
MOV v1, {0, 0.5, 1, 0};
XPD t, v0, v1;
```

DO SIDENESS  
TEST

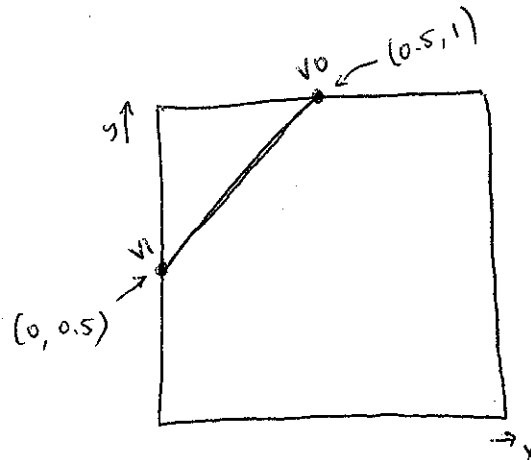
```
{
MOV uv_temp, uv;
MOV uv_temp.z, 1;
DP3 t, t, uv_temp;
```

```
CMP result.color, t, 0, 1;
```

```
END
```

This question is a little tricky!

You were supposed to notice that the cross-product XPD computes the line equation between (0.5, 1) and (0, 0.5) and the DP3 does a "sidedness" test.



28. [5] Using the header for the fragment program above, write the remainder of the program that would draw a circle of radius 0.25 (in texture space) at the center of the screen. The circle should be red {1,0,0,0}, the rest of the background in black {0,0,0,0}.

```
TEMP t;
TEMP r; # define radius
```

Equation of circle:

$$(x-a)^2 + (y-b)^2 < r^2$$

where (a,b) is center of circle

```
SUB t, uv, 0.5; # t = x-0.5, y-0.5, 0, 0
MOV t.zw, 0; #
```

```
DP3 t, t, t; # t = (x-0.5)^2 + (y-0.5)^2
```

```
MUL r, 0.25, 0.25; # r = 0.25^2
```

```
SUB t, t, r; # t = (x-0.5)^2 + (y-0.5)^2 - 0.25^2
```

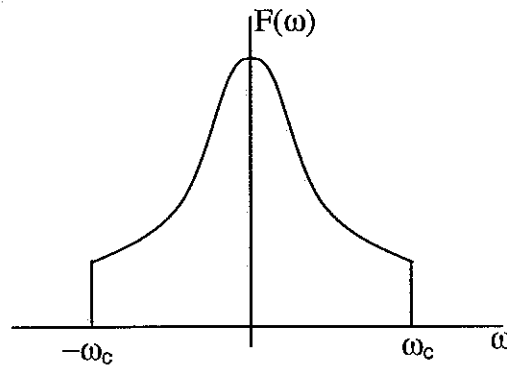
```
CMP result.color, t, {1, 0, 0, 0}, 0; # shade red inside circle
# black outside
```

```
END
```

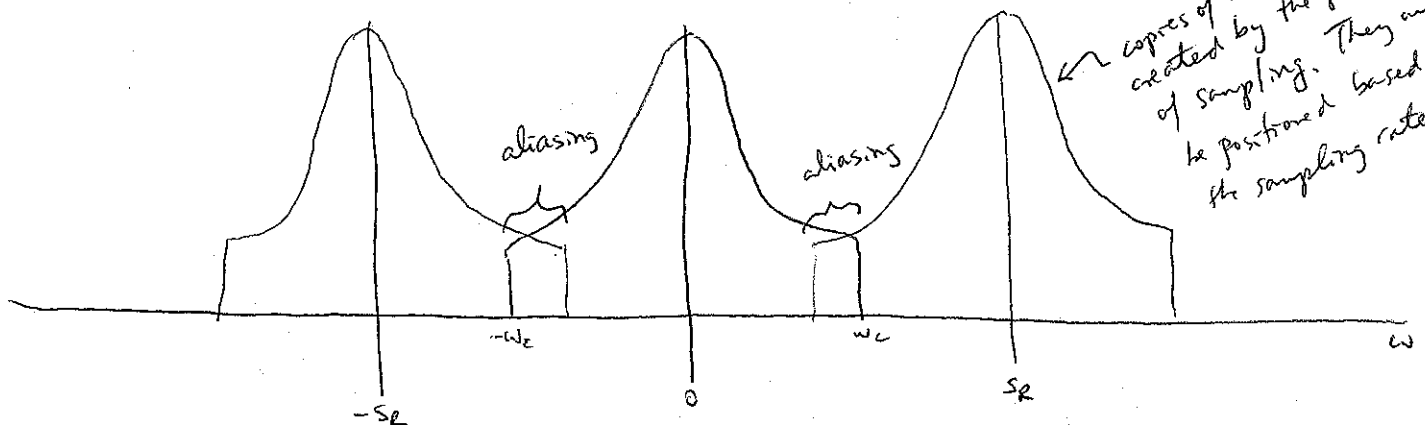
29. [4] In class, we talked about the different ways we could represent the spatially-continuous theoretical texture signal. What are the four different techniques we talked about? Note that some of them could be approximations of the theoretical signal.

- 1) EXPLICITLY - with a mathematical expression
- 2) IMPLICITLY - with a program, as in a shader
- 3) IMPLICITLY - with functional combination of objects whose properties are mathematically defined (circles, lines, spheres, etc) ← vector graphics
- 4) APPROXIMATELY - using a discrete texture to approximate the continuous signal

30. [6] Suppose you are given a continuous signal that has the frequency spectrum shown below. Show what the frequency spectrum would look like when the signal has been under-sampled and is aliasing [5]. Identify the source of the aliasing in your diagram [1].



When it is under-sampled and is aliasing, then the spectrum will overlap with each repetition. Sampling rate  $S_R < 2\omega_c$



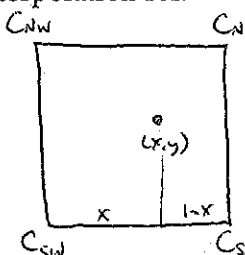
31. [3] A signal  $f(t)$  passed through a linear system (such as a filter) can be modeled as a convolution of  $f(t)$  by the impulse response of the system  $g(t)$  to result in the output  $h(t)$ . What does the Fourier transform of the output  $h(t)$ ?

$$h(t) = f(t) * g(t) \xrightarrow{F} H(\omega) = F(\omega)G(\omega)$$

32. [10] Describe why you need the Texture Sampling Filter (TSF) and the Screen-space Sampling Filter (SSF) that we discussed in class.

We assume the user starts off with a continuous texture signal (the theoretical texture signal) which he wants to map onto the surface. However, it is difficult to present a theoretical signal like that to the hardware for rendering. Typically, this signal has to be discretized (sampled) to be used. To avoid aliasing during this sampling process, the signal is filtered by what we called the Texture Sampling Filter (TSF). When the discrete signal has been reconstructed into a continuous signal on the hardware, it will be resampled again by the screen sampler. Therefore, we need to filter it again to avoid aliasing. This is done by the SSF. In graphics hardware, this is typically done using mipmapping.

33. [5] Assume that you are doing texture mapping and interpolation samples using a bilinear interpolation. Suppose that the corner colors are given by  $C_{NE}$ ,  $C_{SE}$ ,  $C_{SW}$ ,  $C_{NW}$ , and the projected point is given in local cell coordinates  $x$  and  $y$  with the origin being at the south-west (SW) corner and the north-east (NE) corner is 1, 1. What is the bilinear interpolation formula?



$$f(x,y) = (1-y)((1-x)C_{SW} + xC_{SE}) + y((1-x)C_{NW} + xC_{NE})$$

34. [2] What are the two kinds of reconstruction filters available in modern graphics hardware?

nearest neighbor (GL\_NEAREST)

bilinear interpolation (GL\_LINEAR)

35. [10] Explain how S3TC compression works and show how it achieves 6:1 compression. In other words, describe the algorithm in enough detail to show how many bits are used in the S3TC representation and what they are used for.

S3TC works on the assumption that colors in a  $4 \times 4$  pixel block will lie along a major axis in RGB color space. So first, the image is divided into  $4 \times 4$  blocks. For each block, the major axis is computed. This axis is encoded by a line segment, with the endpoints representing 2 different colors, let's call them A and B. If we store A, B, we can compute an additional 2 colors in between  $\frac{1}{3}A + \frac{2}{3}B$  and  $\frac{2}{3}A + \frac{1}{3}B$ . This gives us a "palette" of 4 colors, which can be used to color the  $4 \times 4$  block and should approximate the original colors in the block. We also need to encode at each pixel which of the 4 colors we will use, which requires 2 bits/pixel. So we have colors A & B to store, which are encoded at 16 bits each (for a total of 32 bits) and the labels at each pixel amount to  $2 \times 16 \text{ pixels} = 32 \text{ bits}$ . So each  $4 \times 4$  block is encoded to 64 bits.

The original block size was  $3 \text{ Bytes} \times 16 \text{ pixels} = 48 \text{ Bytes}$ .

S3TC encoded block is 8 B. Therefore compression ratio is  $\frac{48}{8} = 6:1$

36. [5] Why can't JPEG be used for texture compression for real-time rendering?

There are several reasons why JPEG can't currently be used in real-time rendering:

- 1) slow decompression - we need very fast decompression
- 2) variable length encoding - JPEG does run-length encoding on the output.

This violates our constant-time or random access requirement for Real Time rendering.

37. [4] Given the following formulas for the Fourier transform and inverse transform:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} dt$$

Prove the shift property of Fourier transforms:

$$f(t-a) \xrightarrow{FT} F(\omega) e^{-j\omega a}$$

$$\begin{aligned} \int_{-\infty}^{\infty} f(t-a) e^{-j\omega t} dt &\Rightarrow \begin{array}{l} \text{change variable} \\ u = t-a \\ du = dt \\ t = u+a \end{array} \Rightarrow \int_{-\infty}^{\infty} f(u) e^{-j\omega(u+a)} du \\ &= \int_{-\infty}^{\infty} f(u) e^{-j\omega u} e^{-j\omega a} du \\ &= e^{-j\omega a} \int_{-\infty}^{\infty} f(u) e^{-j\omega u} du \\ &= \boxed{e^{-j\omega a} F(\omega)} \end{aligned}$$

38. [6] What are hard and soft shadows?

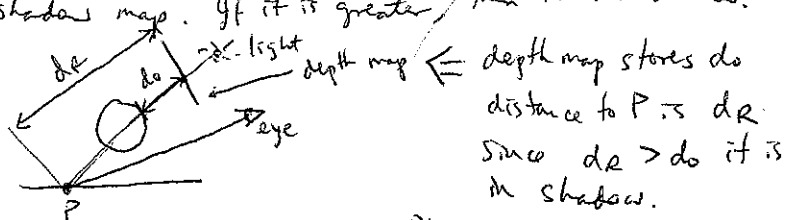
Hard shadows are shadows cast by a point light source. The decision is binary: things are either in shadow or in light. In other words, the surface points can either see the light source or not.

Soft shadows are shadows cast from area or extended light sources. The shadows are now softer because you could now have points that can see the entire light source, points that do not see any of it, and points that see only part of it. It is no longer a binary situation with a hard boundary.

39. [10] The two principal hard shadow algorithms we talked about in class are shadow maps and shadow volumes. Explain how each of them works, as if you are describing them to someone so that they can implement them [6]. It might be helpful to draw diagrams. Also, what are the advantages/disadvantages of each [4]?

Shadow maps

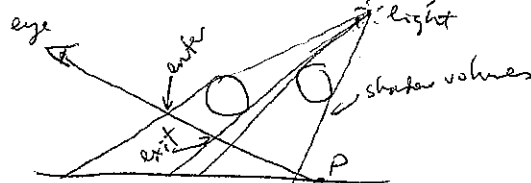
- 1) Take a depth map from the p.o.v. of the light source.
- 2) During rendering, project point to be shaded P into the depth map. Compare the distance from P to the light source to that stored in the shadow map. If it is greater then it is in shadow. Else in light.



PROBLEM: Artifacts due to quantization of the depth map.

Shadow Volumes

Count the number of times the eye rays enter and exit shadow volumes. If for every time it enters a shadow volume, it has exit, then it is in light. Else shadow.



PROBLEM: High rasterization cost to draw the shadow volumes

40. [2] What is the source of the artifacts in shadow maps?

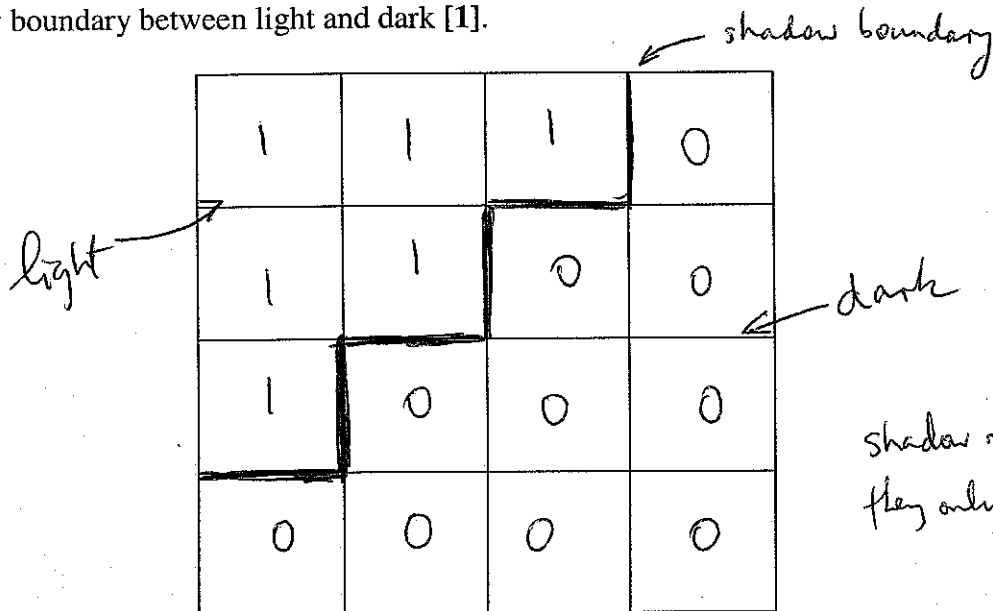
One of the major sources of artifacts in shadow maps is that the depth map (or shadow map) is discrete and not continuous. Since you are trying to represent continuous occluders with a sampled representation, you will have aliasing. Also the quantization of the values in the depth map (due to precision) can cause some problem with self-shadowing.

41. [8] Percentage-Closer Filtering is a technique used to improve the quality of the shadow map algorithm in high-end rendering systems, such as those used by Pixar. Suppose you were given the depth map shown below.

100.1	100.2	100.3	100.4	101	25
100	100.1	100.2	100.1	25	25
99.9	99.8	99.9	25	25	25
98.7	98.9	25	25	25	25
97	25	25	25	25	25
25	25	25	25	25	25

Your task is to shade the inner 4 x 4 region (the cells in white) for a polygon who spans the entire region and whose depth is 30. To present your solution, use the following notation: write a 1.0 in the regions where the polygon is fully illuminated, a 0.0 for the parts where it will be fully dark. For the regions where the shaded value is in between 0 and 1, write it either as a fraction or a decimal.

First shade the result you would get using standard shadow maps [2]. Highlight the shadow boundary between light and dark [1].



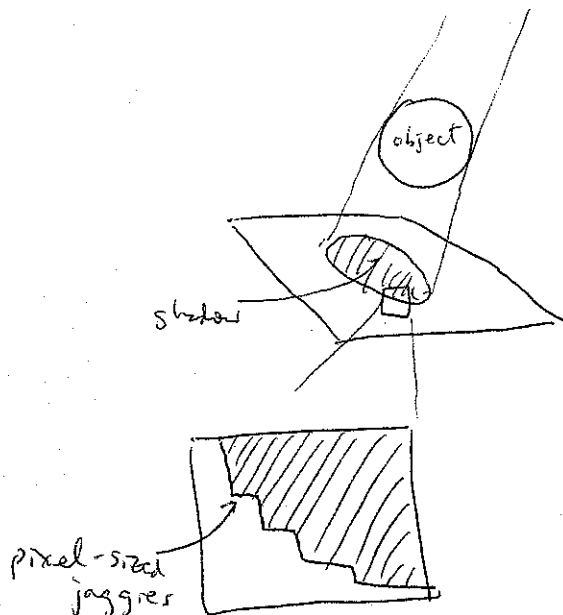
Now shade the 4x4 region of the surface with Percentage Closer Filtering [5]. For every sample, use the four nearest depth samples for comparison.

1	1	1	0.75	0.25
1	1	0.75	0.25	0
1	0.75	0.25	0	0
0.75	0.75	0	0	0
0.25	0	0	0	0

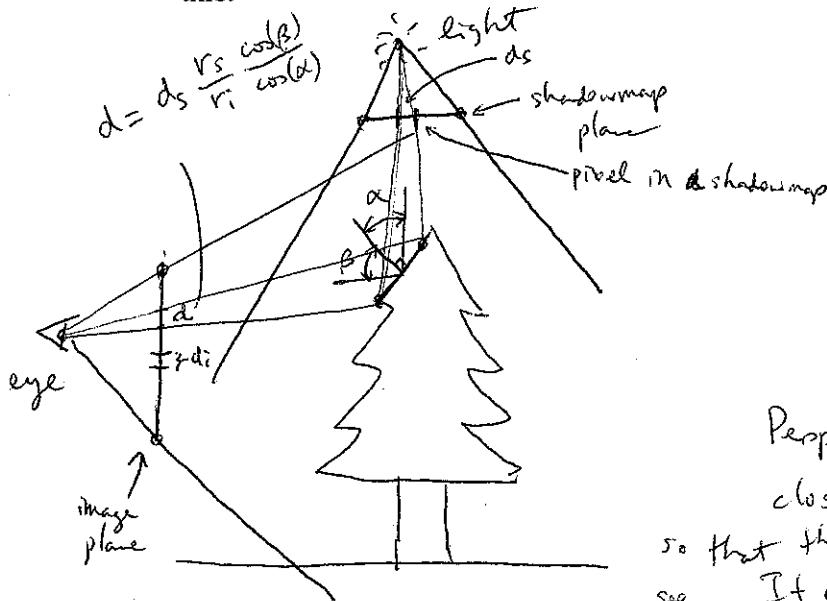
Note that we look at a 2x2 neighborhood of samples, so we have to draw our new grid that is offset from the new one.

42. [5] Hard shadows can contain infinitely-high frequencies, which means that unless they are filtered properly they will exhibit aliasing artifacts when sampled for the screen. We talked about some of the artifacts of shadow maps. However, shadow volumes are not filtered either. What do the aliasing artifacts of shadow volumes look like? Describe them.

The aliasing of shadow volumes shows up as pixel-sized "jaggies" at the boundary between light and dark. Ideally, we would like a slight blur (anti-aliasing) but with standard shadow volumes that is not the case.



43. [5] In class, we talked about the differences between projective aliasing and perspective aliasing when it comes to shadow maps. What are the differences between the two and how do they come about? It might help to draw some diagrams to explain this.



Projective aliasing occurs when the ~~surface~~ surfaces are tilted with respect to the shadow map plane. For example, in the figure when  $\alpha \rightarrow 90^\circ$  then a single shadow map pixel will be "projected" into a larger # of pixels visible by the eye, so the artifacts will be visible. Eg. when  $\frac{\cos(\beta)}{\cos(\alpha)} > 1$ .

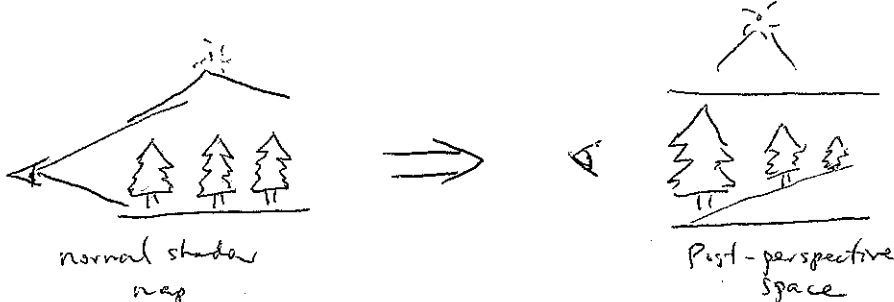
Perspective aliasing occurs when an object is closer to the eye than to the light source so that the quantization from the shadow map can be seen. It can also happen when the resolution of the shadow map is low. Mathematically, it means that  $\frac{d_s r_s}{d_i r_i} > 1$ .

↳ from Stamminger et al.

44. [5] Describe how perspective shadow maps strive to reduce the artifacts of shadow maps. Diagrams might help.

Perspective shadow maps try to reduce artifacts by computing the shadow map in post-projective space. In other words first the scene is ~~proj~~ transformed by the eye's projection matrix (to make objects near the eye bigger) and then the shadow map is generated. One of the problems is that this transformation will also transform the light source.

Basic idea:



45. [2] What are the two largest manufacturers of consumer-level graphics hardware (e.g. the GPU's we've been talking about in the class)?

NVIDIA, ATI (AMD)