

CPSC 314

Quiz 2

2pm, Tuesday November, 15 2005

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name: _____

Student Number: _____

Question 1	/ 13
Question 2	/ 10
Question 3	/ 47
Question 4	/ 30
TOTAL	/ 100

1. Perspective

- (a) (7 points) The standard perspective warp transformation matrix (below) uses the Z coordinate as the view direction. Write the perspective warp matrix if the view direction is along the X axis.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{d}{d-\alpha} & \frac{-\alpha d}{d-\alpha} \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}$$

$$\text{answer} = \begin{pmatrix} \frac{d}{d-\alpha} & 0 & 0 & \frac{-\alpha d}{d-\alpha} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{d} & 0 & 0 & 0 \end{pmatrix}$$

- (b) (6 points) What is the geometric implication of setting $\alpha = 0$ in the perspective warp matrix? Our matrix becomes

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}$$

So any vector $(x, y, z, 1)^T$ multiplied by this matrix becomes $(\frac{xd}{z}, \frac{yd}{z}, d, 1)^T$. All points are mapped onto $z = d$ plane and so depth information is lost.

2. (10 points) A scene contains the triangle $T = (P_1, P_2, P_3)$ with $P_1 = (0, 1, 0), P_2 = (0, 2, 0), P_3 = (0, 2, 2)$ and with texture (u, v) coordinates at the vertices defined as $(1, 0), (1, 1),$ and $(0, 0)$ respectively. Compute the texture coordinates at the point $P = (0, 1.5, 0.5)$ on the triangle. Explain the stages of your computation.

We use the 3 areas formula to compute barycentric coordinates at P . $A_{P_1P_2P_3} = 1, A_{P_1P_2P} = 1, A_{P_1P_3P} = 1, A_{P_2P_3P} = 1. a1 = A_{P_2P_3P}/A_{P_1P_2P_3} = 0.5, a2 = A_{P_1P_3P}/A_{P_1P_2P_3} = 0.25, a3 = A_{P_1P_2P}/A_{P_1P_2P_3} = 0.25.$ Given these coordinates, the texture coordinates at P are $a1 * (1, 0) + a2 * (1, 1) + a3 * (0, 0) = (0.75, 0.25)$

3. Scan Conversion and Clipping

A circle is defined by a center $C = (C_x, C_y)$ and a radius R (assume that C_x, C_y and R are integer).

- (a) (5 points) Write an implicit equation of the circle.

$$(x - C_x)^2 + (y - C_y)^2 = R^2$$

or

$$\|P - C\|^2 = R^2$$

where P is a point in 2D.

- (b) (6 points) Write a parametric equation of the circle.

$$x(2\pi t) = R * \cos(t)$$

$$y(2\pi t) = R * \sin(t)$$

Where $t \in [0, 1]$.

- (c) (12 points) Write the pseudocode for rasterizing an arc on the circle where $C_x \leq x \leq C_x + (R/2)$ and $C_y - R \leq y \leq C_y - R/2$. Make your code as efficient as possible. (Hint: start with a floating point algorithm first and convert it to integer if you can).

As with line drawing we use decision variable d and set it to be the value of the implicit function f at the midpoint between the next two possible pixels. Remember that if $f(x, y) < 0$ then we are inside the circle and if $f(x, y) > 0$ then we are outside the circle. For simplicity, assume the circle is at the origin (which makes $f(x, y) = x^2 + y^2 - R^2$) and translate the coordinates by (C_x, C_y) at draw time. This gives us the following pseudocode:

```

y = -R // set the starting point at the bottom of the circle (0, -R)
x = 0
d = f(x + 1, y + 0.5) // initialize the decision variable
SetPixel(C_x + x, C_y + y) // draw the first point
while((y ≥ R/2)&&(x ≤ (R/2)))
    if(d < 0){ // if the midpoint is in the circle go East
        d = f(x + 2, y + 0.5) // update the decision variable
    }
    else{ // if the midpoint is outside the circle go NorthEast
        d = f(x + 2, y + 1.5) // update the decision variable
        y = y + 1
    }
    x = x + 1
    SetPixel(C_x + x, C_y + y) // draw the new point
}

```

To construct an integer algorithm, we need only to change our operations with d . We do this by expanding and simplifying our terms involving f . First, when $y = -R$ and $x = 0$, then our initialization $d = f(x + 1, y + 0.5)$ becomes $d = 1^2 + (0.5 - R)^2 + R^2$ or more simply $d = 1.25 - R$. Because we're only going to be adding integers to the decision variable we can drop the fractional term yielding the integer operation $d = 1 - R$. Next, instead of computing f for every iteration we must compute an integer increment to d . If $d < 0$ then this increment is $f(x + 2, y + 0.5) - f(x + 1, y + 0.5)$ which, when expanded simplifies to $2x + 3$. If $d > 0$ then this increment is $f(x + 2, y + 1.5) - f(x + 1, y + 0.5)$ which simplifies to $2 * (x - y) + 5$ when expanded. We assemble these integer operations into the following pseudocode

```

y = -R // set the starting point at the bottom of the circle (0, -R)
x = 0
d = 1 - R // initialize the decision variable
SetPixel(Cx + x, Cy + y)
while((y ≥ R/2)&&(x ≤ (R/2)))
    if(d < 0){ // if the midpoint is in the circle go E
        d+ = 2 * x + 3 // increment the decision variable
    }
    else{ // if the midpoint is outside the circle go NE
        d+ = 2 * (x - y) + 5 // increment the decision variable
        y = y + 1
    }
    x = x + 1
    SetPixel(Cx + x, Cy + y)
}

```

- (d) (12 points) Write the pseudocode for scan converting the circle (including its interior).

```

y = Cy - R
x = Cx - R
while(y < Cy + R)
    while(x < Cx + R){
        if((x - Cx)2 + (y - Cy)2 ≤ R2)
            SetPixel(x, y)
        }
        x = x + 1
    }
    y = y + 1
}

```

- (e) (12 points) Write the pseudocode for clipping a line P_1, P_2 against the circle.

If you insert the parametric equation for the line into the implicit equation for the circle, we obtain the quadratic equation

$$\|P_1 + t(P_2 - P_1) - C\|^2 = R^2$$

If we define $d = (P_2 - P_1)$ and $\Delta = (P_1 - C)$, we can rewrite this as.

$$\|td - \Delta\|^2 - R^2 = 0$$

This expands into

$$(d_x^2 + d_y^2)t^2 + 2(d_x\Delta_x + d_y\Delta_y)t + (\Delta_x^2 + \Delta_y^2) - R^2 = 0$$

which we simplify using vector notation into a familiar quadratic form $at^2 + 2bt + c$

$$\|d\|^2t^2 + 2(d \cdot \Delta)t + (\|\Delta\|^2 - R^2) = 0$$

If this equation has no real roots, then the line doesn't intersect the circle. If it has one real root, then the line is tangent to the circle. We're only interested if it has two real roots. Taking this into account yields the following pseudocode to compute the clip points with the circle

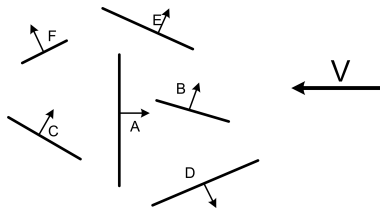
```

[t1, t2] = getroots(P1, P2, C, R); // compute the intersection points with the circle
if(t1 == NULL){ // return if there aren't two roots
    return
}
if(f(P1) < 0){ // Is P1 in the circle?
    if(f(P2) > 0){ // Is P2 out of the circle?
        if(0 < t1 < 1){
            P2 = P1 + t1 * (P2 - P1) // clip P2
        }
        else{
            P2 = P1 + t2 * (P2 - P1) // clip P2
        }
    }
}
elseif(f(P2) < 0){ // Is P2 in the circle?
    if(0 < t1 < 1){
        P1 = P1 + t1 * (P2 - P1) // clip P1
    }
    else{
        P1 = P1 + t2 * (P2 - P1) // clip P1
    }
}
else{ // both of our points are outside circle
    TEMP = P1 + t1 * (P2 - P1) // clip P1
    P2 = P1 + t2 * (P2 - P1) // clip P2
    P1 = TEMP
}
// function computes the roots NULL if there aren't two real roots
function[t1, t2] = getroots(P1, P2, C, R){
    d = P2 - P1
    Δ = P1 - C
    a = ||d||2 // part A of quadratic equation
    b = d · Δ // part B of quadratic equation
    c = (||Δ||2 - R2) // part C of quadratic equation
    δ = b2 - ac
    if(δ > 0){ // this check ensures there are two real solutions
        t1 =  $\frac{-b - \sqrt{\delta}}{c}$  // compute the first root
        t2 =  $\frac{-b + \sqrt{\delta}}{c}$  // compute the second root
        return[t1, t2]
    }
    return NULL
}

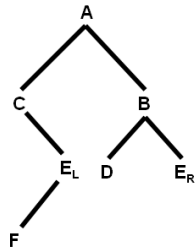
```

4. BSP Trees

- (a) (10 points) Construct the BSP tree for the segments shown below (use alphabetical insertion order for the segments, when possible). Use the convention where the right subtree (child) is located on the side that the normal points to. Show your work.



The line of A splits E into E_L and E_R .



- (b) (10 points) Given the view direction V as shown above, describe the complete traversal order of your BSP tree during rendering.

$$(A-)A(A+)$$

$$(C-)C(C+)A(B-)B(B+)$$

$$C(E_L-)E_L(E_L+)ADBE_R$$

$$CFE_LADBE_R$$

- (c) A scene consists of several non-intersecting polygons whose plane equations are of the form $Ax + D = 0$ (i.e. $B = C = 0$).

- i. (5 points) Given a view direction along the X axis, is it always possible to sort the polygons for drawing, without splitting any of them? Explain.

Yes. The polygons are all on parallel planes so there can be no intersection with which to split.

- ii. (5 points) Given a view direction along the vector $(1, 1, 1)$, is it always possible to sort the polygons for drawing, without splitting any of them? Explain.

Yes. BSPs are view-independent. Changing the view does not effect the construction.