ECE433: PC GRAPHICS

Winter Semester 2021-2022

Homework 2

Group of students #18

Κουτσούκης Νικόλαος

## Exercise 1

In computer graphics, the Liang–Barsky algorithm is a line clipping algorithm. The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clip window. With these intersections it knows which portion of the line should be drawn. The idea of the Liang–Barsky clipping algorithm is to do as much testing as possible before computing line intersections.

Parametric definition of a line:

x = x1 + u\*Δx

y = y1 + u\*Δy

Δx = (x2-x1), Δy = (y2-y1), 0 <= u <= 1

p1 = -Δx, q1 = x1 - xmin

p2 = Δx, q2 = xmax – x1

p3 = -Δy, q3 = y1 - ymin

p4 = -Δy, q4 = ymax – y1

Pseudo code:

for (each line segment to be clipped)

Pre-calculate pk and qk;

if (P1 = P0) then line is degenerate so clip as a point;

else

begin

u1 = 0;

u2 = 1;

for (each candidate intersection with a clip edge)

if pk = 0 then {Ignore edges parallel to line}

begin

calculate u; {of line and clip edge intersection}

use sign of pk to categorize as Pstart or Pend;

if Pstart then u1 = max(0,u1,u);

If Pend then u2 = min(1,u2,u);

end

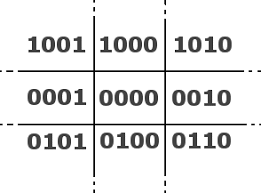
if u1 > u2 then return nil

else return P(u1) and P(u2) as true clip intersections

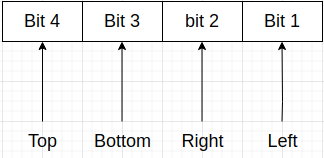
end

## Exercise 2

The Cohen–Sutherland algorithm is a computer-graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are visible in the central region of interest (the viewport).



The edge of every line contains a 4-bit value which determines its position.



The rules of the algorithm:

Classify p0, p1 using region codes c0, c1(4-bit value)

If c0 ᴧ c1 != 0, trivially reject

If c0 ˅ c1 , trivially accept

Otherwise reduce to trivial cases by splitting into two segments

Pseudo code:

ComputeOutCode(x0, y0, outcode0)

ComputeOutCode(x1, y1, outcode1)

repeat

check for trivial reject or trivial accept pick the point that is outside the clip rectangle

if TOP then

x = x0 + (x1 – x0) \* (ymax – y0)/(y1 – y0); y = ymax;

else if BOTTOM then

x = x0 + (x1 – x0) \* (ymin – y0)/(y1 – y0); y = ymin;

else if RIGHT then

y = y0 + (y1 – y0) \* (xmax – x0)/(x1 – x0); x = xmax;

else if LEFT then

y = y0 + (y1 – y0) \* (xmin – x0)/(x1 – x0); x = xmin;

end {calculate the line segment}

if (x0, y0 is the outer point) then

x0 = x; y0 = y; ComputeOutCode(x0, y0, outcode0)

else

x1 = x; y1 = y; ComputeOutCode(x1, y1, outcode1)

end {Subdivide}

until done

## Exercise 3

At first, we check the points P0, P1.

P0 ^ P1 = 0, so we don’t reject the line.

**Step 1:**

|  |  |
| --- | --- |
|  | Because first we check if the point is it on top, that’s means P1 is on top and not on right.  P1 = 1010 → TOP  newX = 4.5  newY = 4  So, P1’ = (4.5, 4)  And now we check P0, P1’ if are inside window. |

**Step 2:**

|  |  |
| --- | --- |
|  | P1’ is inside so we check only the P0.  P0 = 0101 → BOTTOM  newX = 1.5  newY = 1  So, P0’ = (1.5, 1)  And now we check P0’, P1’ if are inside window.  P0’ ˅ P1’ = 0 ,therefore we accept the line (P0’, P1’ are inside) |

## Exercise 4

Δx = 5.5 – 0.5 = 5

Δy = 5 – 0 = 5

**Step 1**

p1 = -5, q1 = 0.5 – 1 = -0.5, r1 = 0.1, u1 = max(r1, 0) = 0.1

**Step 2**

p2 = 5, q2 = 5 - 0.5 = 4.5, r2 = 0.9, u2 = min(r2, 1) = 0.9

**Step 3**

p3 = -5, q3 = 0 – 1 = -1, r3 = 0.2, u1 = max(r3, 0) = 0.2

**Step 4**

p4 = 5, q4 = 4 – 0 = 4 , r4 = 0.8, u2 = min(r4, 1) = 0.8

**Step 5**

u1 = 0.2, u2 = 0.8

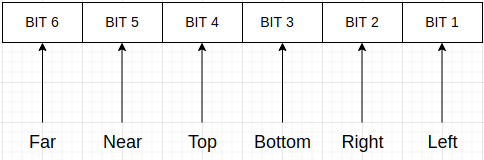
P0’ = (x0 + u1\*Δx, y0 + u1\*Δy) = (1.5, 1)

P1’ = (x0 + u2\*Δx, y0 + u2\*Δy) = (4.5, 4)

|  |
| --- |
|  |

## Exercise 5

Now we use a 6-bit value to handle the near and far plane.



Pseudo code:

ComputeOutCode(x0, y0, z0, outcode0)

ComputeOutCode(x1, y1, z1, outcode1)

repeat

check for trivial reject or trivial accept pick the point that is outside the clip rectangle

if TOP then

x = x0 + (x1 – x0) \* (ymax – y0)/(y1 – y0); y = ymax; z = znew;

else if BOTTOM then

x = x0 + (x1 – x0) \* (ymin – y0)/(y1 – y0); y = ymin; z = znew;

else if RIGHT then

y = y0 + (y1 – y0) \* (xmax – x0)/(x1 – x0); x = xmax; z = znew;

else if LEFT then

y = y0 + (y1 – y0) \* (xmin – x0)/(x1 – x0); x = xmin; z = znew;

else if NEAR then

y = ynew, x = xnew, z = zmax;

else if FAR then

y = ynew, x = xnew, z = zmin;

end {calculate the line segment}

if (x0, y0, z0 is the outer point) then

x0 = x; y0 = y; z0 = z; ComputeOutCode(x0, y0, z0, outcode0)

else

x1 = x; y1 = y; z0 = z; ComputeOutCode(x1, y1, z1,outcode1)

end {Subdivide}

until done

## Exercise 6

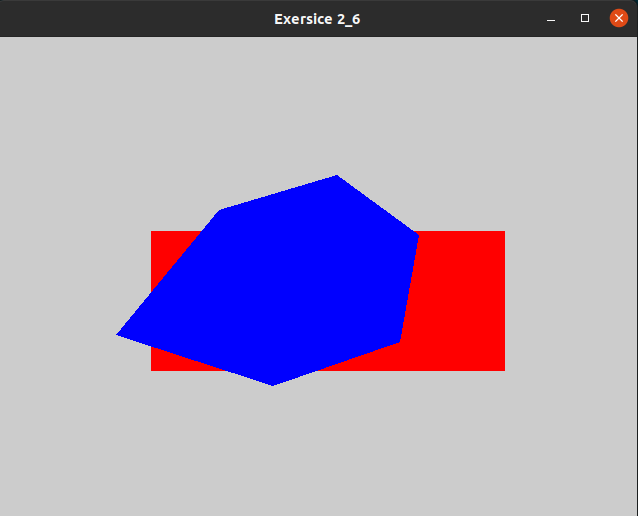
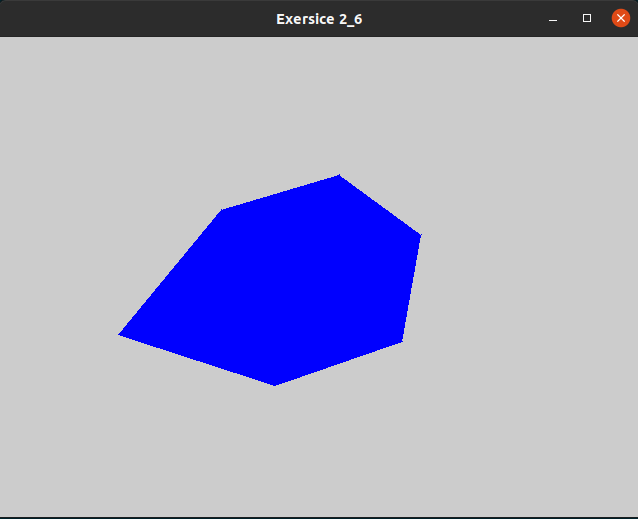
Implementation of a polygon cutting program.

First, with the opening of the application, we have the option to select the edges of the polygon (max 10 edges) by left clicking.

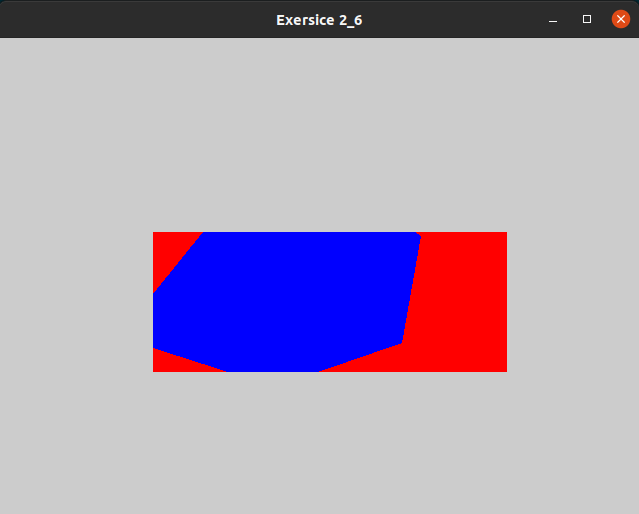




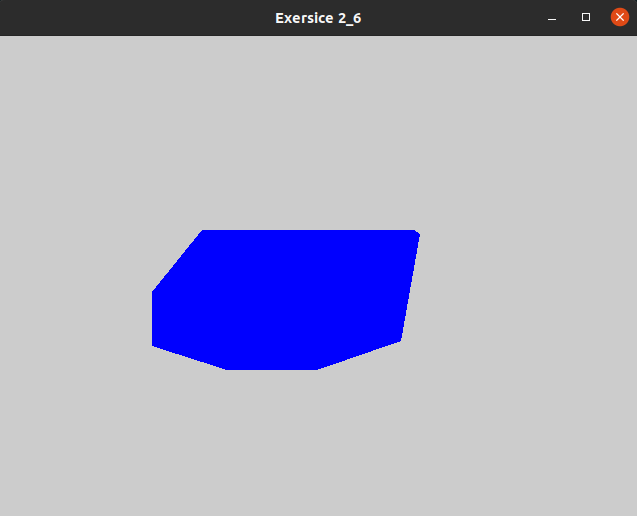
## Then, by pressing the button, ‘F1’ the polygon is colored blue, and we have the option by dragging the left click to select the cutting polygon, colored red.



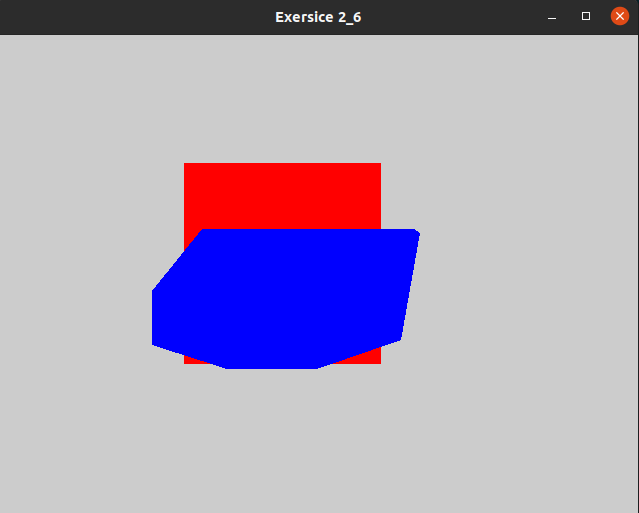
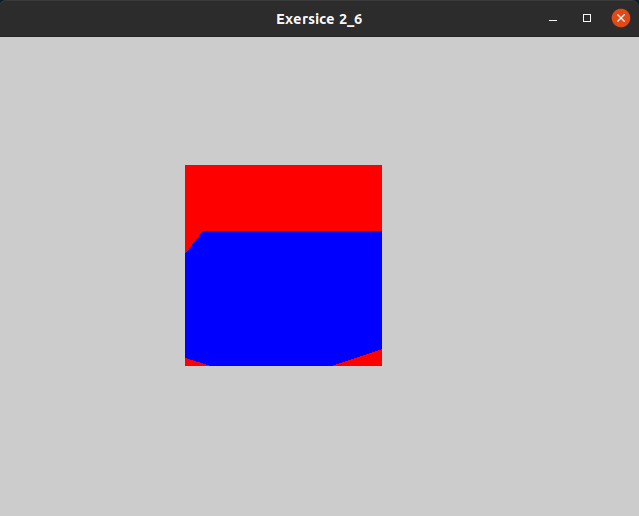
After selecting the cutting polygon, by pressing the ‘C’ or ‘c’ button, the polygon is cut.



Also, we have the choice, any time we want, of the appearance or not of the cutting polygon (red polygon), by pressing the spacebar button.



Finally, after the completion of the cut we can continue the same process by selecting a new cut polygon in the already modified polygon or we can right-click which clears the screen so we can start over.



## Exercise 7

Sutherland-Hodgman Clipping Algorithm

Four cases:

1. S inside plane and E inside plane:

Add E to output ,(note, S has already been added).

1. S inside plane and E outside plane:

Find intersection point I, add I to output.

1. S outside plane and E outside plane:

Add nothing.

1. S outside plane and E inside plane:

Find intersection point I, add I to output, followed by E.

**step 1:**

|  |  |
| --- | --- |
|  | First, we look for the left side:  AB case 2 → save I₁  BC case 4 → save I₂ and C  CD, DE, EF, FA case 1 → save D, E, F, A |

**step 2:**

|  |  |
| --- | --- |
|  | Now we look for the right side:  AI₁, I₁I₂, I₂C, case 1 → save I₁, I₂, C  CD case 2 → save I₃  DE case 4 → save I₄, E  EF, FA case 1 → save F, A |

**step 3:**

|  |  |
| --- | --- |
| I1  I2  I3  I4  I5  I6  I7  I8  I9  I10 | Bottom side:  AI₁ case 4 → save I₅, I₁  I₁I₂, I₂C, CI₃, I₃I₄ case 1 → save I₂, C, I₃, I₄  I₄E case 2 → save I₆  EF case 4 → save I₇, F  FA case 2 → save I₈ |

**step 4:**

|  |  |
| --- | --- |
|  | Top side:  I₂C case 2 → save I₉  CI₃ case 4 → save I₁₀, I₃  all the others are case 1,  So, our final output is:  I₁, I₂, I₉, I₁₀, I₃, I₄, I₆, I₇, F, I₈, I₅ |

## Exercise 9

HSV -> RGB

C = V × S

H’ = H / 60ᵒ

X = C(1 – |H’ mod 2 - 1|)

(0, 0, 0) if H is undefined

(C, X, 0) if 0 ≤ H′ < 1

(X, C, 0) if 1 ≤ H′ < 2

(R, G, B) = (0, C, X) if 2 ≤ H′ < 3

(0, X, C) if 3 ≤ H′ < 4

(X, 0, C) if 4 ≤ H′ < 5

(C, 0, X) if 5 ≤ H′ < 6

RGB -> HSV

V = max(R, G, B), min = min (R, G, B)

S = (max – min) / max (or S = 0, if V = 0)

0 + (G – B) / (max – min), if max = R

H = 60 × 2 + (B – R) / (max – min), if max = G

4 + (R – G) / (max – min), if max = B

H = H + 360, if H < 0