Computação Gráfica – Ciência da Computação – 6º Período
Prof. Paulo Bressan – Universidade Federal de Alfenas

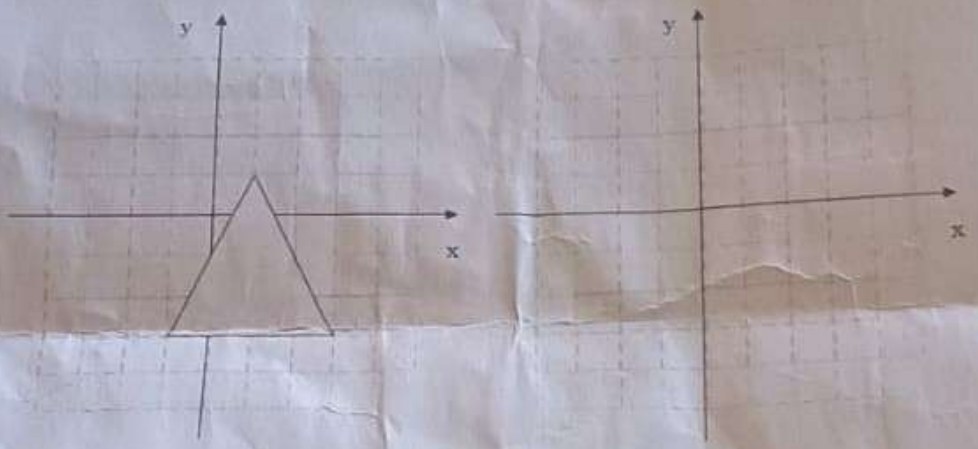
Nome: .....

Avaliação Presencial Especial – 11/Dez/2023

1ª. a) (2,0) Encontre a matriz resultante para a translação de -1 unidades no eixo x e +2 unidades no eixo y, depois um escalonamento de +2 unidades no eixo y e no final uma rotação de +30°:

b) (1,0) Para a figura abaixo, desenhe a casinha na posição correta por meio da matriz resultante do item a).

Informações: cos(30°) = 0,86 e sen(30°) = 0,50



Cada quadrado corresponde a 1 unidade de largura e 1 unidade de altura.

2ª. (2,0) Considerando os pontos $P_1 = (-40, -50)$, $P_2 = (80, -80)$, $P_3 = (60, 100)$, encontre os pontos sobre a curva quadrática de Bézier com passo de 0,2 para t. Desenhe a curva e os pontos em um gráfico.

3ª. (0,5) (Poscomp 2017 - Modificada) No processo de visualização tridimensional, a região do universo que será recortada e projetada sobre o plano de projeção é denominada:

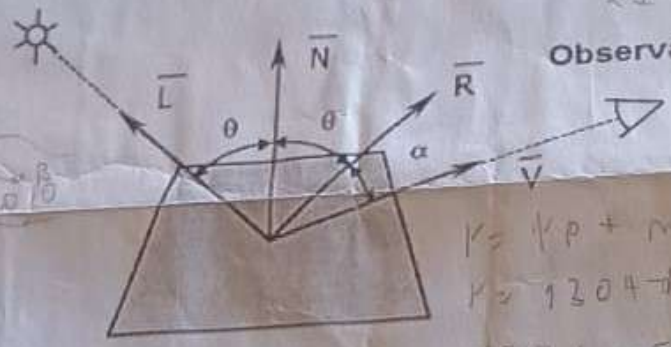
A) Projeção perspectiva.
B) Observador.
C) Sistema de referência da câmera.
D) Volume de visão.
E) Plano de recorte frontal.

4ª. (0,5) (Poscomp 2016 - Modificada) Assinale a alternativa **incorreta** que descreve uma característica de transformações de projeção

A) A projeção perspectiva não preserva ângulos e medidas de objetos.
B) Projeções isométricas são projeções paralelas.
C) Em uma projeção paralela, considera-se que o centro de projeção está a uma distância determinada do plano de projeção.
D) O tamanho da projeção perspectiva de um objeto varia de forma diretamente proporcional a distância desse objeto ao centro de projeção.
E) Uma projeção perspectiva pode ser representada por uma matriz 4x4.

5ª. (2,0) Considerando os valores do plano de projeção como sendo $x_{min} = -20$, $x_{máx} = +80$, $y_{min} = -40$ e $y_{máx} = +100$, descreva os passos para o recorte do segmento formado pelos pontos A = (40, 130) e B = (120, 20) seguindo o algoritmo de Cohen-Sutherland, e indique os pontos finais caso exista alguma parte do segmento dentro do plano de projeção.

6ª. (2,0) (PosComp 2011 - Modificado) Em cenas de computação gráfica, para aumentar o realismo visual, é comum aplicar-se um modelo de iluminação local que calcula as cores nos vértices dos triângulos a partir das propriedades de reflexão do objeto, propriedades geométricas do objeto e propriedades da(s) fonte(s) de luz.



Sobre os modelos de iluminação locais, considere as afirmativas a seguir:

I. A parcela de reflexão especular depende da posição do observador.
II. A parcela difusa ideal de iluminação pode ser aproximada pela lei de Lambert, que estabelece que a reflexão difusa de uma superfície é proporcional ao ângulo entre o vetor normal à superfície e o vetor direção da fonte de luz.
III. A parcela especular pode ser aproximada pelo modelo de Phong, que estabelece que a reflexão especular de uma superfície é proporcional ao cosseno do ângulo entre o vetor normal e o vetor oposto da fonte de luz.
IV. O modelo de sombreamento de Gouraud interpola normais, enquanto o modelo de sombreamento de Phong interpola cores.

Assinale cada afirmativa acima como Falso ou Verdadeiro, justificando o erro nas alternativas falsas.

```
// C++ Program to implement Cohen Sutherland algorithm
// for line clipping.
#include <iostream>
using namespace std;

// Defining region codes
const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Defining x_max, y_max and x_min, y_min for
// clipping rectangle. Since diagonal points are
// enough to define a rectangle
const int x_max = 10;
const int y_max = 8;
const int x_min = 4;
const int y_min = 4;

// Function to compute region code for a point(x, y)
int computeCode(double x, double y) {
    // initialized as being inside
    int code = INSIDE;
    if (x < x_min)      // to the left of rectangle
        code |= LEFT;
    else if (x > x_max) // to the right of rectangle
        code |= RIGHT;
    if (y < y_min)      // below the rectangle
        code |= BOTTOM;
    else if (y > y_max) // above the rectangle
        code |= TOP;
    return code;
}

// Implementing Cohen-Sutherland algorithm
// Clipping a line from P1 = (x2, y2) to P2 = (x2, y2)
bool cohenSutherlandClip(double x1, double y1, double x2, double y2) {
    // Compute region codes for P1, P2
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);

    while (true) {
        if ((code1 == 0) && (code2 == 0)) {
            // If both endpoints lie within rectangle
            return true;
        }
        else if (code1 & code2) {
            // If both endpoints are outside rectangle, in same region
            return false;
        }
        else {
            // Some segment of line lies within the rectangle
            int code_out;
            double x, y;

            // At least one endpoint is outside the  rectangle, pick it.
            if (code1 != 0)
                code_out = code1;
            else
                code_out = code2;

            // Find intersection point; using formulas y = y1 + m * (x - x1),
            // x = x1 + (1 / m) * (y - y1)
            if (code_out & TOP) {
                // point is above the clip rectangle
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
                y = y_max;
            }
            else if (code_out & BOTTOM) {
                // point is below the rectangle
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
                y = y_min;
            }
            else if (code_out & RIGHT) {
                // point is to the right of rectangle
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
                x = x_max;
            }
            else if (code_out & LEFT) {
                // point is to the left of rectangle
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
                x = x_min;
            }
            // Now intersection point x,y is found
            // We replace point outside rectangle by intersection point
            if (code_out == code1) {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1);
            } else {
                x2 = x;
                y2 = y;
                code2 = computeCode(x2, y2);
            }
        }
    }
}
```