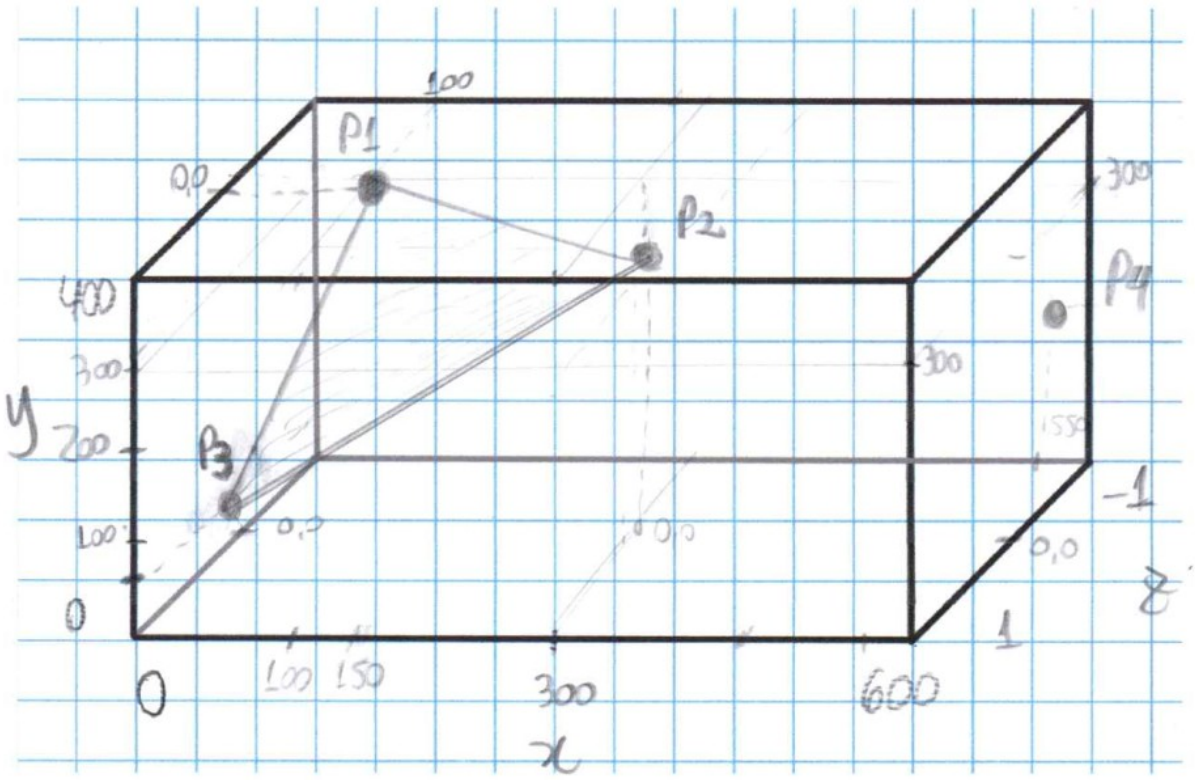
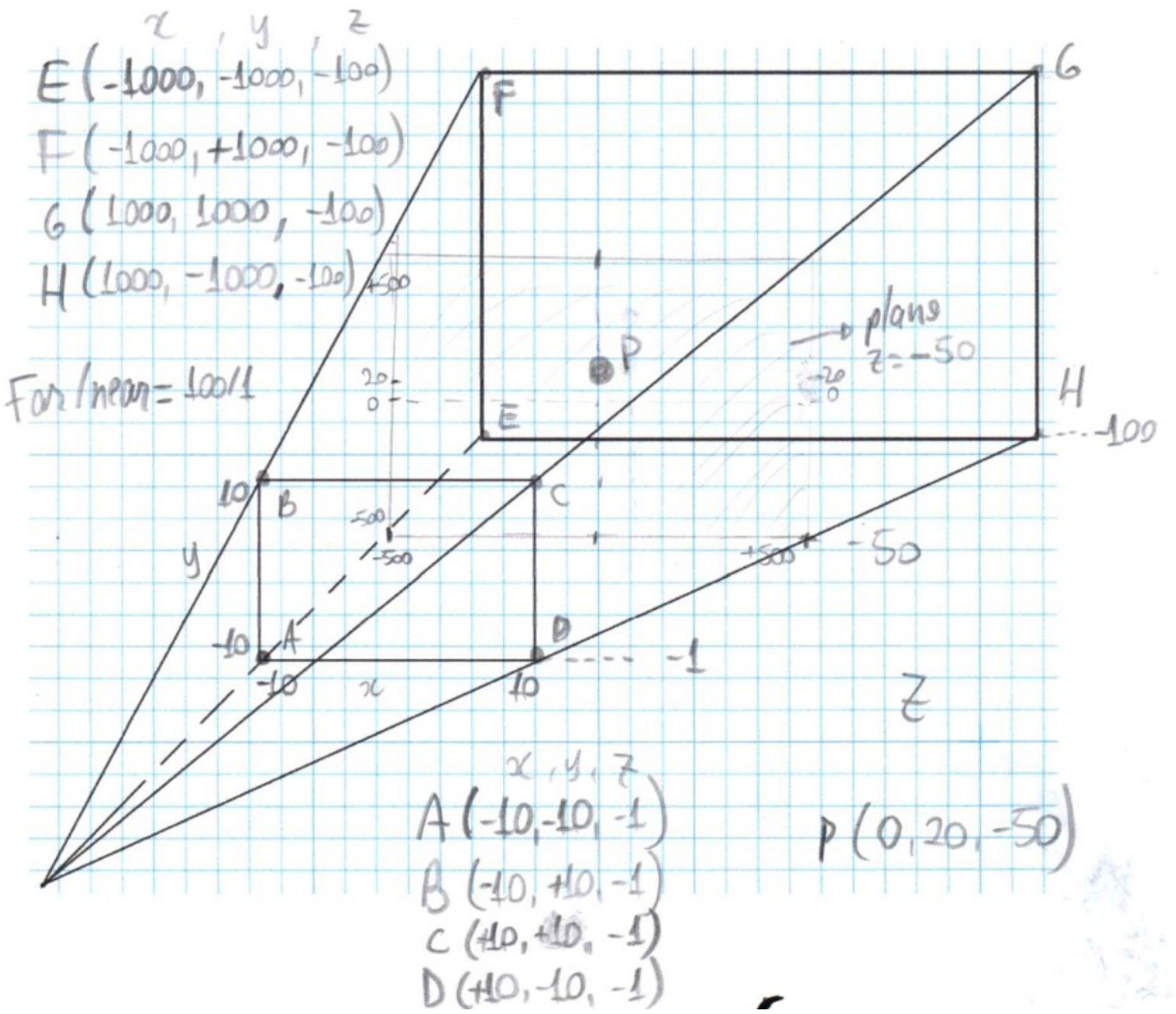
Lista de Exercícios 2: Preparação para prova escrita

Exercícios de preparação para a prova escrita (devem ser feitos à mão). Não precisa entregar.

1. Elabore o desenho de um Viewing Box definido pela seguinte configuração: glOrtho(0,600,0,400,-1, 1); // l,r,b,t,n,f
   a. Posicione e identifique o ponto p1(100,400,0) no viewing box;
   b. Posicione e identifique o ponto p2(300,300,0) no viewing box;
   c. Posicione e identifique o ponto p3(0, 50, 0) no viewing box;
   d. Posicione e identifique o ponto p4(550,200,-1) no viewing box;
   e. Suponha que os pontos acima estivessem definidos dentre as instruções: glBegin(GL_TRIANGLE); …. glEnd(); represente graficamente (no mesmo desenho) a saída gráfica esperada para este programa.



2. Elabore o desenho de um Viewing Frustum definido pela seguinte configuração: glFrustum(-10,10,-10,10,1,100); // l,r,b,t,n,f
   a. Considerando a regra da mão direita e a origem o(0,0,0), identifique, com valores numéricos, todos os vértices delimitadores deste viewing frustum, tanto no plano *near* como no plano *far;*
   b. Posicione (aproximadamente) e identifique o ponto p1(0,20,-50) neste viewing frustum.



3. Descreva, em pseudo-código ou linguagem C, o algoritmo para se obter o efeito de um espiral ao longo do eixo Z. O espiral deverá ter raio *r* e estar centrado na posição 0, 0, *-near*.

```
// supondo near=−10, far=−100
glBegin(GL_LINE_STRIP);
for (t = −10; t >= −100; t−=0.5){ // decremento de t determina segmentos de circ.
      x = r*cos(t);
      y = r*sin(t);
      z = t;      // efeito "zoom out"
      glVertex3f(x, y, z);
      // lembrar que na proj perspectiva z=0 esta na origem do observador
      // por isso, deslocar para zoom out
   }
```

4. Como funciona o algoritmo z-buffer? Descreva seu funcionamento e complemente sua descrição com um pseudo-código para este algoritmo.

   Inicializar matriz z-buffer [x,y] ← -inf // inicializa

   Para cada modelo da cena

   Para cada pixel (x,y) projetado

   profundidade_z ← valor de z do polígono na coordenada [x,y] de tela

   SE (profundidade_z) > (z-buffer [x,y])

   ENTÃO

   z-buffer [x,y] ← profundidade_z // ponto sobrescreve

   cor [x, y] ← cor_pixel (x,y) // cor de saída do modelo na posição (x,y)

5. É preciso instalar o OpenGL para utilização em um computador? Fundamente sua resposta.

   O OpenGL é uma API para computação gráfica, ou seja, existem funcionalidades especificadas pelo consórcio mantenedor (Grupo Khronos) e os fabricantes devem garantir que estas funcionalidades estejam presentes nos produtos comercializados. Sendo assim, os fabricantes de hardware de processamento gráfico devem garantir a implementação das funcionalidades do OpenGL, bem como o seu interfaceamento com sistemas operacionais por meio dos respectivos drivers. Pode-se concluir que não é preciso "instalar" o OpenGL, e sim, utilizar um driver atualizado para utilizar recursos do OpenGL num sistema operacional. A comunicação com a placa gráfica também depende deste sistema operacional utilizado, e, normalmente, as bibliotecas responsáveis por este interfaceamento já se fazem presentes no respectivo S.O. (por exemplo, opengl32 para Windows, libGL para Linux).

6. O que significa sistema de coordenadas NDC?

   Normalized Device Coordinates ou sistema de coordenadas normalizadas do dispositivo; neste sistema, as coordenadas de tela (x,y) estão contidas no domínio [-1,1]. Este sistema é utilizado pelo OpenGL no processamento gráfico.

7. Quais os procedimentos para se transformar coordenadas dos vértices modeladas em um Viewing Frustum (projeção perspectiva) e obter o resultado em NDC?

   Considerando a modelagem em um viewing frustum (projeção perspectiva), deve-se transformar os vértices (x,y,z) para coordenadas homogêneas (x,y,z,w=1). Em seguida, deve-se transformar os vértices por meio da multiplicação pela matriz de projeção. Ou seja, para cada vértice v, tem-se o vértice transformado t(v):

   t(v) = Mproj * v // matriz de projeção perspectiva – resultado (xp,yp,zp,wp)

   Por fim, após a transformação, retorna-se às coordenadas cartesianas (x,y,z) dividindo os valores por wp, obtendo assim:

   xt = yp/wp

   yt = zp/ wp

Os resultados de tela (xt,yt) contidos no intervalo [-1,1] estarão na região visível, enquanto os resultados fora deste intervalo estarão fora dos limites da tela (*clipping*)

8. O que são *shaders*?

   Os shaders são programas que serão processados no hardware de processamento gráfico (GPU). No OpenGL, são codificados na linguagem GLSL (OpenGL Shading Language).

9. Qual as principais características do *vertex shader* e do *fragment shader*?

   Vertex shader: responsável pela transformação para o cálculo da posição correta dos vértices modelados em World Coordinates para a respectiva posição na tela. Ou seja, no vertex shader, aplicam-se as transformações tais como escala, translação, rotação e a projeção. O vertex shader também é responsável por encaminhar demais informações dos vértices (por exemplo, os componentes r,g,b,a) para demais estágios de processamento do OpenGL.

   Fragment shader: responsável pela determinação da cor de cada pixel, também denominado de pixel shader em alguns contextos. Para a determinação da cor de saída, podem estar envolvidos, além de valores r,g,b,a, outros recursos, tais como efeitos de luz, brilhos, constrastes, sombras e texturas.

10. Em que consiste a transformação de translação? Dê um exemplo.

    Tal transformação permite deslocar a posição dos vértices. Aplicam-se fatores de translação (tx,ty,tz) às coordenadas (x,y,z). Ex: dado um ponto p(2,2,0), aplicando-se o fator de translação tx=2, tem-se como ponto resultante: p'(4,2,0).

11. Assinale V ou F:

    a. (F) Com o OpenGL moderno, o programador deve implementar seu próprio algoritmo de *clipping*, diferentemente do OpenGL antigo // o clipping é feito pelo próprio OpenGL no pipeline gráfico

    b. (F) O *vertex shader* pode ser desprezado caso as coordenadas já estejam em NDC // o vertex shader é requerido no OpenGL moderno, pode-se então simplesmente não fazer nenhuma transformação e deixar a saída do vertex shader com os próprios valores de entrada (ou usar matriz identidade na transformação t(v) = M*v)

    c. (F) Para a transformação de translação, poderia ser utilizada uma matriz 3x3 // a translação é uma transformação afim, é impossível, por exemplo, transladar um ponto na origem (0,0,0) por meio de multiplicações por quaisquer valores, ou seja, não existe uma matriz capaz de calcular tal transformação $t(v) = M_{3x3} * (v)$ para transladar este ponto.

    d. (V) O OpenGL não dispõe de recursos nativos para exibir caracteres na tela // de fato, o OpenGL não lida com saída por meio de caracteres nativamente, pode-se utilizar mapeamento de texturas ou definir caracteres com as primitivas usuais

    e. (V) A primitiva GL_POLYGON foi descontinuada no OpenGL moderno // a primitiva de polígono, bem como a de quadriláteros, foram descontinuadas. Com computação gráfica "moderna", recorre-se a triângulos (mais eficiência em cálculos)

f. (V) Pode-se definir um Viewing Frustum utilizando o argumento fovy // os ângulos de campo de visão podem ser utilizados, há relação matemática explícita entre tais ângulos e o viewing frustum, tanto para fov-y e fov-x.