Aula Prática Laboratorial n.º 8, nº 9 e nº 10

Sumário

Tutorial Nate Robins "*texture*". Programa C/OpenGL "labirinto" baseado no template "*Aula8_template.cpp"*

Tutorial Nate Robins "***textures***"

Execute o tutorial "*texture*" de Nate Robins e veja os projectos "*labirinto*" a funcionar.

Projecto "labirinto"

Iniciar o projecto

- Crie um projecto do tipo consola
- Acrescente o ficheiro "*Aula8_template.cpp*" que foi disponibilizado;
- Acrescente o ficheiro "*readjpeg.c*" para ler texturas em formato JPEG;
- Acrescentar os ficheiros "*mdlviewer.cpp*", "*math.cpp*", "*studio_render.cpp*" e "*studio_utils.cpp*" para ler ficheiros com extensão ".mdl" (Half Life 2);
- Teste o exemplo;

Desenhar o labirinto

Como o projecto usa duas *subwindows* é necessário criar *Display-Lists* e *Texturas* para cada uma das *subwindows*.

- Altere a função *desenhaLabirinto()* para desenhar o labirinto baseado na matriz ***mazedata***. O labirinto deve estar centrado em 0,0,0 e ser desenhado com cubos usando a função *desenhaCubo()*.
- A função *desenhaCubo()* só tem uma normal definida. Altere o vector Normais e as chamadas à função *desenhaPoligono(...).*

As *Display-Lists* que contêm o chão e o labirinto já são criadas na função *createDisplayLists(int **janelaID**)* e chamadas nas funções *displayTopSubwindow()* e *displayNavigateSubwindow().*

Animar o objecto

- Usar a função *Timer()* para animar o objecto. Os dados estão na estrutura modelo.objecto.{pos.{x,y,z},vel,dir}.
- Alterar a função *detectaColisao(GLfloat **nx**,GLfloat **nz**)* para detectar se o objecto colidiu com algum cubo e impedir a sua movimentação.

Animar a câmara

- Usar a função *setNavigateSubwindowCamera(camera_t ***cam**, *objecto_t **obj**)* para mover a câmara com o objecto use duas abordagens:
  - A câmara está atrás do objecto a olhar para o centro do objecto
  - A câmara está no objecto a olhar para a frente
- Permitir que a câmara possa ser alterada com o rato (altera estado.camera.dir_lat e estado.camera.dir_long) para isso use coordenadas esféricas para colocar a câmara (ver observações)
  - Na função *mouseNavigateSubwindow()* quando carrega na tecla esquerda guarde as coordenadas do rato e registe a função *motionNavigateSubwindow()* como *glutMotionFunc()*
  - Na função *motionNavigateSubwindow()* calcule a diferença entre as coordenadas do rato e transforme essa diferença em latitude e longitude

Colocar "Ajudas" sobre a janela

Altere as funções *desenhaBussola(int **width**, int **height**)* e *desenhaModeloDir(objecto_t **obj**,int **width**, int **height**)* para apresentar as informações no canto das subwindows seguindo estes passos:

- Redefinir o *viewport* e a projecção (neste caso 2D)
- Alterar definições *OpenGL*, neste caso permitir transparências e desligar iluminação. Como queremos que tudo seja desenhado também desligamos o teste de profundidade. Ver observações sobre transparências.

Colocar Texturas nos cubos

Para colocar texturas é necessário antes de se especificar um vértice especificar uma coordenada de textura com a instrução *glTexCoord2f(**s**,**t**).* A textura utilizada para os cubos tem 16 faces diferentes e como a imagem é mapeada entre 0 e 1, cada face tem

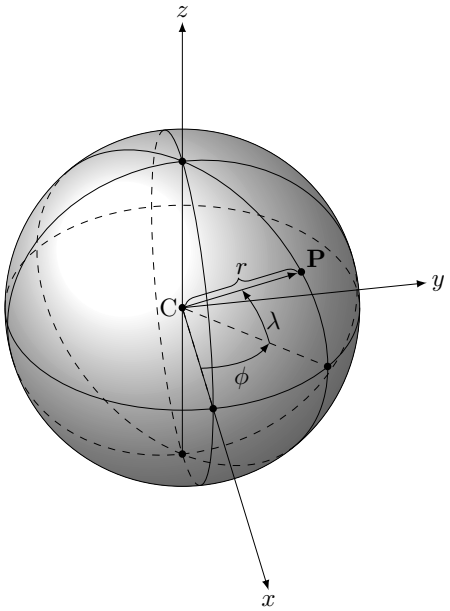
uma área de 0.25, exemplo podemos mapear uma face com as seguintes coordenadas (0,0); (0.25,0); (0.25,0.25) e (0,0.25);

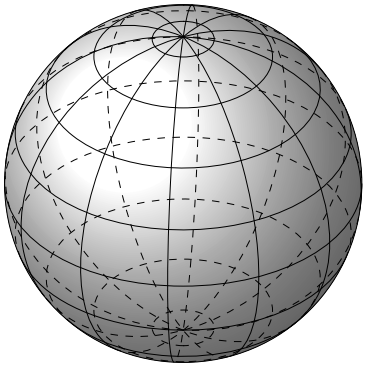
As texturas são carregadas a partir de ficheiros BMP (cubo) e JPEG (chão) na função *createTextures(GLuint **texID**[]).* Cada textura tem um identificador OpenGL que é guardado no vector passado como parâmetro, este identificador é usado na função *glBindTexture(**GL_TEXTURE_2D**, **ID**)* sempre que queremos usar essa textura. Quando não queremos usar texturas podemos desligá-las com as instruções *glDisable(**GL_TEXTURE2D**);* ou *glBindTexture(**GL_TEXTURE_2D**, **NULL**);*

Observações

Coordenadas esféricas

As coordenadas esféricas são definidas a partir de dois ângulos: a latitude λ (círculos paralelos ao equador) e a longitude ϕ (círculos que passam pelos pólos).





$$\begin{cases} P_x = C_x + r\cos\phi\cos\lambda \\ P_y = C_y + r\sin\phi\cos\lambda \\ P_z = C_z + r\sin\lambda \end{cases} \quad \text{com,} \quad \begin{array}{rcl} -\pi/2 & \leq \lambda \leq & \pi/2 \\ 0 & \leq \phi < & 2\pi \end{array}$$

Transparências

As transparências devem ser os últimos objectos a ser desenhados e idealmente do mais distante para o mais próximo da câmara. Deve ser desligada a escrita no *buffer* de profundidade, mas permitir a sua leitura com a instrução *glDepthMask(**GL_FALSE**)*. Existe um exemplo simples na função *desenhaAngVisao(camera_t ***cam**)*

Animação do Homer

Para carregar o modelo usa-se a função:

*mdlviewer_init(char ***modelname**, StudioModel &**tempmodel**)*

Para desenhar o modelo usa-se a função

*mdlviewer_display(StudioModel &**tempmodel**)*

Para alterar a animação do modelo usam-se as funções/métodos

*mdlviewer_nextsequence(StudioModel &**tempmodel**)*

*mdlviewer_prevsequence(StudioModel &**tempmodel**)*

*modelo.SetSequence(int **valor**);*

modelo.GetSequence();

Game Mode

Nos jogos desenvolvidos com o *GLUT*, deve-se usar o "*Game Mode*". Este tipo de janela é mais eficiente que as janelas normais *Windows* e pode ser especificada a resolução, a profundidade do *buffer* de cores e a taxa de refrescamento.

Para criar uma janela em "*Game Mode*" deve-se usar as seguintes instruções:

```
glutGameModeString("1024x768:32@75"); // 1º teste
if (glutGameModeGet(GLUT_GAME_MODE_POSSIBLE))
  glutEnterGameMode();
else
{
  glutGameModeString("800x600:32@60"); // 2º teste
  if (glutGameModeGet(GLUT_GAME_MODE_POSSIBLE))
    glutEnterGameMode();
  else // Cria Janela Normal
  {
    glutInitWindowPosition(10, 10);
    glutInitWindowSize(800, 600);
    if ((estado.mainWindow=glutCreateWindow("Labirinto")) == GL_FALSE)
      exit(1);
  }
}
```

Para esconder a consola inserir esta linha no código

```
#pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
```

Instruções usadas

*glutGet(GLenum **state**)*

Pede ao *GLUT* uma das suas variáveis de estado (inteiros)

`GLUT_WINDOW_X`, `GLUT_WINDOW_Y` , `GLUT_WINDOW_WIDTH`, `GLUT_WINDOW_HEIGHT` , `GLUT_WINDOW_CURSOR`, `GLUT_SCREEN_WIDTH`, `GLUT_SCREEN_HEIGHT`, `GLUT_DISPLAY_MODE_POSSIBLE`, `etc.`

*void glBlendFunc(GLenum **sfactor**, GLenum **dfactor**)*

sfactor Especifica como a imagem origem (existente no frame buffer) é tratada e pode ser:

`GL_ZERO`, `GL_ONE`, `GL_DST_COLOR`, `GL_ONE_MINUS_DST_COLOR`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_DST_ALPHA`, `GL_ONE_MINUS_DST_ALPHA`, ou `GL_SRC_ALPHA_SATURATE`.

dfactor Especifica como a imagem criada é tratada e pode ser:

`GL_ZERO`, `GL_ONE`, `GL_SRC_COLOR`, `GL_ONE_MINUS_SRC_COLOR`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_DST_ALPHA`, ou `GL_ONE_MINUS_DST_ALPHA`.