Ficha Prática n.º 6

Importante:

A ficha deve ser realizada em **grupo (2-3 alunos)**, seguindo as seguintes normas:

- A constituição dos grupos deve ser a mesma da ficha anterior;
- A submissão da ficha prática deve ser através do **Canvas**, em *assignment* próprio, e deverá consistir num ficheiro **ZIP** com uma pasta contendo o **código-fonte**.
- A data-limite para entrega da ficha prática é **02 de janeiro de 2022 às 23:55**;
- A apresentação da ficha prática decorrerá nos dias 04 e 06 de janeiro 2022, no horário das aulas práticas;
- No dia da apresentação, **TODOS** os elementos do grupo deverão estar presentes. Os elementos ausentes serão classificados com 0 valores;
- A apresentação e discussão poderá ser realizada individualmente.

1. Projeto Labirinto

1.1. Crie uma cópia do projeto da aula anterior, onde já tenha o Visual Studio Code configurado corretamente. Deste modo, não é necessário configurar novamente o IDE.

1.2. Abra a pasta do novo projeto no Visual Studio Code e apague todos os ficheiros excepto a pasta *.vscode*

1.3. Faça download do ficheiro "template_ficha06.zip" e coloque o conteúdo na pasta do novo projeto. Será necessário atualizar o ficheiro *tasks.json* para que compile também o ficheiro *glm.c*, por exemplo:

```
"args": [
    "-g",
    "${file}",
    "${fileDirname}/glm.c",
```

1.4. **(3 valores)** Desenhe o Labirinto:

- Como o projeto 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()*

1.5. **(2 valores)** Anime o Personagem:

- Usar a função *timer()* para animar o personagem. Os dados estão na estrutura *modelo.objeto.{pos.{x,y,z},vel,dir}*
- Adicione código na função *detectaColisao(GLfloat **nx**, GLfloat **nz**)* para detetar se o objeto colidiu com algum cubo e impedir a sua movimentação

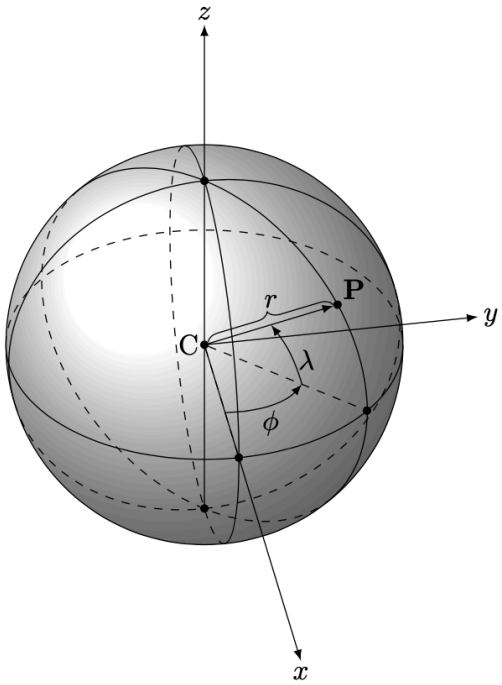
1.6. **(6 valores)** Anime a Câmara:

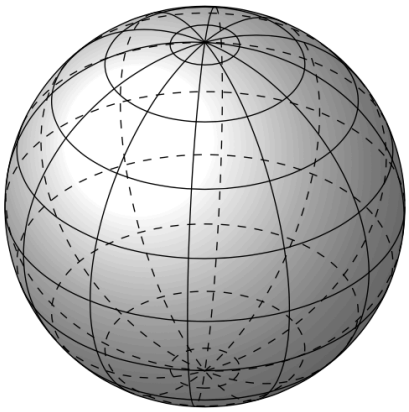
- Usar a função *setNavigateSubwindowCamera(Camera ***cam**, Objeto **obj**)* para mover a câmara com o personagem. Use duas abordagens, sendo possível alterar com a tecla F2:
  1. A câmara está atrás do objeto a olhar para o centro do objeto
  2. A câmara está no objeto a olhar para a frente

- 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:
  - Na função *mouseNavigateSubwindow()* quando carrega no botão direito do rato guarde as coordenadas do rato e registe a função *motionNavigateSubwindow()* como callback da função *glutMotionFunc()*
  - Na função *motionNavigateSubwindow()* calcule a diferença entre as coordenadas do rato e transforme essa diferença em latitude e longitude

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).





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

1.7. **(2 valores)** Coloque "ajudas" na 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 projeção (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.

Transparências:

As transparências devem ser os últimos objetos 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 ***cam**).*

1.8. **(5 valores)** Coloque texturas nos cubos:

- Para colocar texturas é necessário que, antes de se especificar um vértice, se especifique 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
- Por exemplo, podemos mapear uma face com as seguintes coordenadas (0,0); (0.25,0); (0.25,0.25); (0,0.25);
- As texturas são carregadas a partir de ficheiros PPM 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 a instrução *glDisable(**GL_TEXTURE2D**);*

1.9. **(2 valores)** Experimente o 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("800x600:32@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\"")
```

NOTA: Em Mac poderá ser necessário executar o programa no Terminal para visualizar o menu de ajuda.

Funções

*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.