Laboratório de Aplicações com Interface Gráfica
Aulas Práticas

MIEIC – 2014/2015

Trabalho Prático 2 – Aperfeiçoamento das Técnicas de Utilização do OpenGL

1.Introdução

O objetivo deste trabalho é introduzir novas técnicas gráficas, como display lists, evaluators e animação, e algumas mais avançadas, como sejam os *shaders* baseados em GLSL (OpenGLShading Language). Propõe-se assim a implementação de algumas funcionalidades em código, que possam depois ser exploradas através de uma extensão à linguagem ANF, e à criação de uma cena que as utilize. Este documento descreve as funcionalidades pretendidas, bem como a extensão proposta. Apesar de não ser obrigatório, recomenda-se a utilização/extensão do parser ANF, realizado no TP1, para suportar as novas funcionalidades solicitadas neste enunciado.

Funcionalidades pretendidas

1 Display lists
   * Implementar o suporte para a criação de uma display list em nós do *scene graph* devidamente identificados no ANF com um novo atributo *displaylist* (ver extensões ao ANF abaixo). Um nó identificado desta forma, assim como os seus descendentes, devem ser armazenados em display lists, numa primeira passagem; nas passagens seguintes, deve ser invocada a *display list* correspondente.

2 Evaluators
   * (Re)crie uma classe Plane de forma a gerar, utilizando evaluators, um plano de dimensões 1 x 1 unidades, assente em XZ, centrado na origem e com a face visível apontando para +Y. Os evaluators devem gerar também as coordenadas de textura para o plano, variando linearmente de (0,0) a (1,1), e os vértices devem possuir normais corretamente atribuídas. O número de divisões deve ser especificado no construtor da classe. Com estes evaluators, criar uma primitiva “plane” na linguagem ANF.
   * Estender os evaluators de forma a poderem representar superfícies de grau 1, 2 ou 3 nas duas direções U e V. Com base nestes evaluators, criar uma nova primitiva “patch” a incluir na linguagem ANF.
   * Criar em código ou usando as extensões ao ANF propostas abaixo um novo objeto que corresponde a um veículo voador que inclua pelo menos uma superfície não-plana gerada utilizando a primitiva "patch".

3 Animação
   * Implementar um conjunto de classes para o suporte para animações.
     a) Implementar a classe Animation como classe base para aplicar animações a um objeto.
     b) Criar a classe LinearAnimation, derivada de Animation, para trajetórias lineares, que permita definir uma animação caracterizada por um vetor de Pontos de Controlo e tempo de duração em segundos.

     Exemplo:
     Pontos de Controlo = {(0,0,0), (1,0,0), (1,1,0)}
     Tempo= 10 s

     O objeto em movimento deve alterar a sua orientação horizontal (x,z) de acordo com a trajetória, de modo a corrigir a direção quando muda de segmento de reta (ou seja, um movimento de "helicóptero").

c) Criar a classe CircularAnimation, derivada de Animation, para trajetórias circulares, que permita definir uma animação caracterizada pelo centro e raio de circunferência, ângulo inicial e ângulo de rotação, e tempo de duração em segundos.

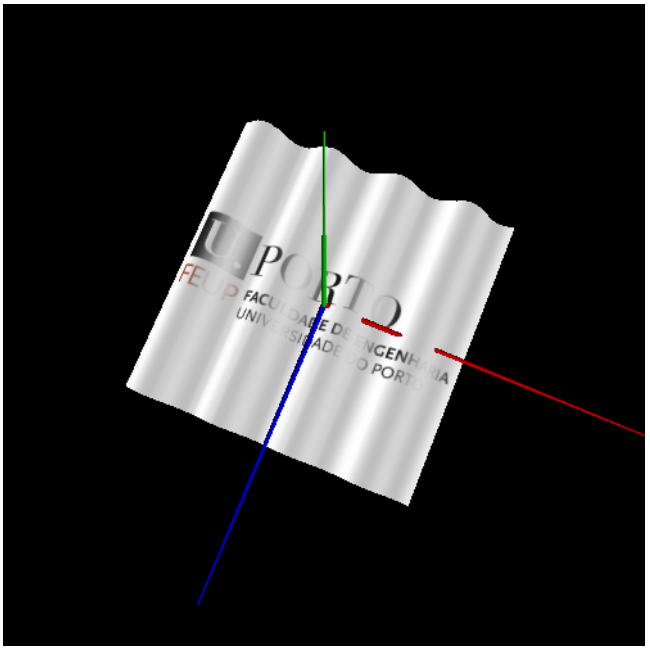
Exemplo:
Centro = (10, 10, 10)
Raio = 5
Ângulo Inicial = 40º
Ângulo de rotação = 20º
Tempo= 20 s

O objeto em movimento deve alterar a sua orientação horizontal (x,z) de acordo com a trajetória.

* Implementar (em código ou usando a extensão proposta à ANF) uma animação para o veículo que inclua, no seu trajeto, pelo menos dois segmentos de reta, recorrendo para tal à classe LinearAnimation, e um segmento circular usando CircularAnimation. O objeto mantém a sua horizontalidade (ou seja, mantém-se paralelo ao plano XZ), apenas podendo rodar em torno do seu eixo vertical, de forma a manter uma orientação coerente com a direção e sentido do seu movimento (como p.ex. um helicóptero).

4 Shaders

* Criar uma nova primitiva Flag, baseada na classe Plane criada anteriormente, para representar uma bandeira a ondular na direção do eixo dos XX, recorrendo a uma textura de cor e um par de shaders (vertex e fragment) - ver imagens e vídeo abaixo.
  a) O vertex shader deve ser aplicado ao plano e alterar as coordenadas de cada vértice de forma a que a sua coordenada Y (altura) varie segundo uma forma sinusoidal controlada pelo valor da coordenada de textura *s*, e essa forma deve deslocar-se ao longo do plano em função do tempo (recebido através da variável time). Ou seja, de forma simbólica, ***newVertex.y= sin(...coord.s...time...)***. Sugere-se que numa primeira versão o tempo seja ignorado, e seja representada uma versão estática da bandeira ondulada.
  b) O fragment shader deve aplicar a textura de cor sobre a bandeira
  c) Deve ser disponibilizado na interface o parâmetro *wind*, para controlar a velocidade de ondulação (e indiretamente o número de ondas: valores elevados representam não só maior velocidade, mas também maior número de ondas, ver vídeo abaixo)



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Textura de Cor	Resultado video com vento constante video com vento alterado interativamente

Requisitos da cena

Deve ser criada uma cena que utilize as funcionalidades referidas acima, nomeadamente:

* Utilização de display lists para pelo menos uma parte da cena.
* Instanciação de uma bandeira;
* Instanciação e animação do veículo animado de acordo com as indicações acima (preferencialmente sobrevoando e/ou pousando/levantando sobre a água).

Nota Importante: Apesar de não ser obrigatório, sugere-se a utilização/extensão do parser ANF realizado no TP1 para suportar as novas funcionalidades solicitadas neste enunciado.

Notas sobre a avaliação do trabalho:

Composição dos Grupos: Os trabalhos devem ser efetuados em grupos de dois estudantes. Em caso de impossibilidade (p.ex. por falta de paridade numa turma), deve ser discutida com o docente a melhor alternativa.
Avaliação do Trabalho de Grupo: A avaliação será feita em aula prática, numa apresentação de cada grupo ao docente respetivo.
Avaliação Individual: Na prova de avaliação individual, serão pedidas várias funcionalidades adicionais, a implementar sobre o código original desenvolvido em trabalho de grupo.
Avaliação do Trabalho: Média aritmética das duas avaliações anteriores.
De acordo com a formulação constante na ficha de disciplina, a avaliação deste trabalho conta para a classificação final com um peso de:

80% * 30% = 24%.

O enunciado incorpora, em cada alínea, a sua classificação máxima, correspondendo esta a um ótimo desenvolvimento, de acordo com os critérios seguintes, e que cumpra com todas as funcionalidades enunciadas. Sem perda da criatividade desejada num trabalho deste tipo, não serão contabilizados, para efeitos de avaliação, quaisquer desenvolvimentos além dos que são pedidos.
Para efeitos de avaliação do trabalho e tendo em atenção as cotações mencionadas anteriormente, serão considerados os seguintes critérios:

* Estruturação e eficiência do software (3 valores);
* Aspeto geral e criatividade (2 valores);
* Utilização das display lists (3 valores);
* Utilização de evaluators (4 valores);
* Animação (4 valores).
* Uso de shaders (4 valores)

Planeamento do Trabalho:

* Semana 1 (início em 20/10/2014): Display lists
* Semana 2 (início em 27/10/2014): Evaluators e animação
* Semana 3 (início em 03/11/2014): Semana de interrupção
* Semana 4 (início em 10/11/2014): shaders (entrega no final da semana, dia 16/11/2014)
* Semana 5 (início em 17/11/2014): avaliação dos trabalhos de grupo durante as aulas práticas
* **Avaliação prática individual**: 21/11/2014

Entrega:

* Data limite de entrega do trabalho completo: 16/11/2014
* Por via eletrónica/moodle (instruções a divulgar).

Sugestão de extensão à Linguagem ANF

A linguagem ANF encontra-se definida no questionário do trabalho prático 1. Nesta secção são apresentadas as extensões ao formato ANF de modo a poder comportar as funcionalidades descritas neste enunciado.

Ao ser lido e interpretado por uma aplicação gráfica, um ficheiro em linguagem ANF deve ser verificado em termos de sintaxe, devendo a aplicação gerar mensagens de erro ou avisos, identificando eventuais erros encontrados ou situações anómalas ou indesejáveis.

NOTA: Todos os identificadores (de tags, atributos...) devem ser escritos exclusivamente com letras minúsculas.

Na descrição abaixo, os símbolos utilizados têm o seguinte significado:

ii: valor inteiro
ff: valor em vírgula-flutuante
ss: string
ee: caracter "x" ou "y" ou "z", especificando um eixo
tt: valor Booleano na forma "true" ou "false"

Segue-se uma listagem representativa da sintaxe pretendida, relativa às extensões à linguagem ANF. As tags / atributos acrescentados encontram-se escritos a vermelho. A cinzento encontram-se

elementos definidos na versão original da linguagem ANF, usados para melhor contextualizar as alterações.

```
<anf>
      …
      <!-- informacao de animacao -->
      <animations>
      <!-- pode não existir qualquer nó "animation" se a cena não tiver animações --
>
      <!-- span é o tempo, em segundos, que a animação deve demorar *
      <!-- nesta versão do formato ANF, type pode ter o valor "linear" ou "circular"
-->
            <animation id="ss" span="ff" type="linear">
                  <controlpoint xx="ff" yy="ff" zz="ff" />
                  ...
            </animation>
            <animation id="ss" span="ff" type="circular" center="ff ff ff"
radius="ff" startang="ff" rotang="ff" />
      </animations>

      <!-- informacao do grafo; "rootid" identifica o no de raiz -->
      <graph rootid="ss" >

            <!-- tem de existir, pelo menos, um bloco "node" -->
            <!-- o parâmetro displaylist (booleano) é opcional -->
            <!-- se este parâmetro não estiver declarado assume-se o valor false --
      >
            <node id="ss" displaylist="tt">
                  …
                  <!-- referência ao bloco de animação por cada
            "node". nó opcional  -->
                  <animationref id="ss" />

                  <primitives>

                        <!– Nova primitiva: plano, gerado por evaluator -->
                        <!– ex: <plane parts="5" /> um plano de dimensões 1 x 1
            unidades -->
                        <!- assente em XZ, centrado na origem e com a face visível
            apontando para +Y -->
                        <!- com divisão em cinco partes por eixo -->
                        <plane parts="ii" />

                        <!– Nova primitiva: patch, gerada por evaluator -->
                        <!– parâmetros: -->
                        <!– order: ordem, pode ser 1,2,3-->
                        <!– partsU: divisão em partes no domínio U a ser usada para o
            cálculo do evaluator -->
                        <!– partsV: divisão em partes no domínio V a ser usada para o
            cálculo do evaluator -->
                        <!– compute: assume 1 de 3 valores possíveis: "point", "line",
            "fill" -->
                        <!– o número de pontos de controlo dentro da primitiva patch é
            (ordem+1)^2 -->
                        <patch order="ii" partsU="ii" partsV="ii" compute="ss">
                          <controlpoint x="ff" y="ff" z="ff" />
                          …
```

```
                    </patch>

                    <!—Nova primitiva: corresponde a um veículo voador. Inclui
          pelo menos uma superfície não-plana gerada utilizando evaluators -->
                    <vehicle />

                    <!— Nova primitiva: bandeira baseada em shaders -->
                    <!- parâmetros: -->
                    <!— texture: ficheiro jpg com a textura que deve ser
          visualizada sobre a bandeira -->
                    <flag texture="ss" />
               </primitives>
          </node >
          …
     </graph>
</anf>
```