Universidade do Minho

Departamento de Informática

Trabalho Prático

Computação Gráfica

André Nunes A85635

Eduardo Teixeira A84707

Vasco Marques A83614

Grupo 9

Fase 3 – Curves, Cubic Surfaces and VBOs

02 de Maio de 2021

Conteúdo

Capítulo 1

Breve Descrição do Enunciado

Nesta terceira fase do trabalho é-nos proposto criar um novo tipo de modelo baseado em Bezier patches, transformando este num ficheiro de pontos que o nosso engine consiga interpretar. Mudamos também a nossa forma de desenhar, passando agora a utilizar VBO's. No lado da engine foram adicionadas novas transformções, as translações e rotações com a nova variável de tempo. Estes novos métodos permitirão a implementação de curvas, superfícies cúbicas e a leitura de um ficheiro patch que contém os pontos para gerar as imagens. Será então criado um sistema solar, tal como na fase anterior, mas de forma dinâmica.

Capítulo 2

Bezier Patches

Para a elaboração dos patches de Bezier, já era fornecido ficheiro .patch, onde estavam os pontos de controlo. Dados os pontos de controlo, a função do generator era ler todos os dados fornecidos no ficheiro, e com eles, construir a matriz com os pontos de controlo, matriz esta que é posteriormente usada para utilizar na fórmula fornecida nas aulas práticas. Convém salientar que usámos matrizes para efetuar os cálculos, e ainda elaboramos uma função para a multiplicação das mesmas. O comando que utilizamos para a elaboração do ficheiro .3d é o seguinte. **Gerar caixa:**

```
$./engine ../scenes/box.xml
```

Capítulo 3

Engine

3.1 Transformações

Com base nas esttruturas anteriormente desenvolvidas nas fases anteriores e com vista nos objetivos desta srgiu a necessidade de armazenar outro tipo de informação. Assim foram adicionadas duas variáveis **time** e **pontos**. Para diferenciar transformações estáticas ou dinâmicas a variável **time** assume o valor igual a zero ou superior a zero, respectivamente.

```
struct transformacao {
        float matriz[4][4];
        float time;
        std::vector<float> * pontos;
};
```

Figura 3.1: Estrutura Transformações.

3.1.1 Rotation

Na transformação de **rotate** passamos a guardar os valores do vetor de rotação na matriz e o **time**. Relativamente à implementação anterior, apenas difere o calculo do ângulo que é feito com a seguinte expressão.

```
float angle = to_radial(glutGet(GLUT_ELAPSED_TIME) * 360.f / t->time);
```

Figura 3.2: Expressão usada para calcular o angulo em cada rotação.

3.1.2 Translate

Na transformação de translação, passamos a guardar o **time** e os **pontos**. Para o cálculo da matriz de transformação é usada a função "calculacatmull", que vai de encontro com a que foi desenvolvida nas aulas práticas no âmbito do guião 8.

3.2 VBOS e Arquitetura

Nesta fase do projeto, foi-nos exigida a implementação de VBOS. Assim, foi necessário criar uma nova arquitetura capaz de suportar tais exigências.

3.2.1 Model

No que toca ao **Model** foi necessario criar uma variável para desenhar os mesmos com VBOS. Esta variável para o buffer do glut é denominada de **vertexBuffer**.

```
struct model {
    std::vector<VERTICE> * pontos;
    std::vector<TRANSFORMACAO> * transformacoes;
    GLuint vertexBuffer[1];
};
```

Figura 3.3: Estrutura do Model.

3.2.2 Scene

A estrutura de **Scene** manteve-se praticamente inalterada com a excessão dos métodos de desenho que tiveram que ser ajustados para incorporar os VBOS.

```
struct scene {
    std::vector<MODEL> * models;

};
```

Figura 3.4: Estrutura da Scene.

3.2.3 Novo parser de xml

Nesta fase o parser de XML também foi adaptado para que permitisse também os novos formatos de translate e rotate, sem deixar de parte as funcionalidades que este possuía anteriormente.

3.3 Resultados

Os resultados não são a representação do trabalho que foi feito, o xml é lido e é analisado ao ponto de criarmos models com pontos e as transformações associadas a cada um deles. Porém a dificuldade em tratarmos os dados com vbos foi maior o que não nos possibilita sequer apresentar um esboço do esforço e do trabalho.

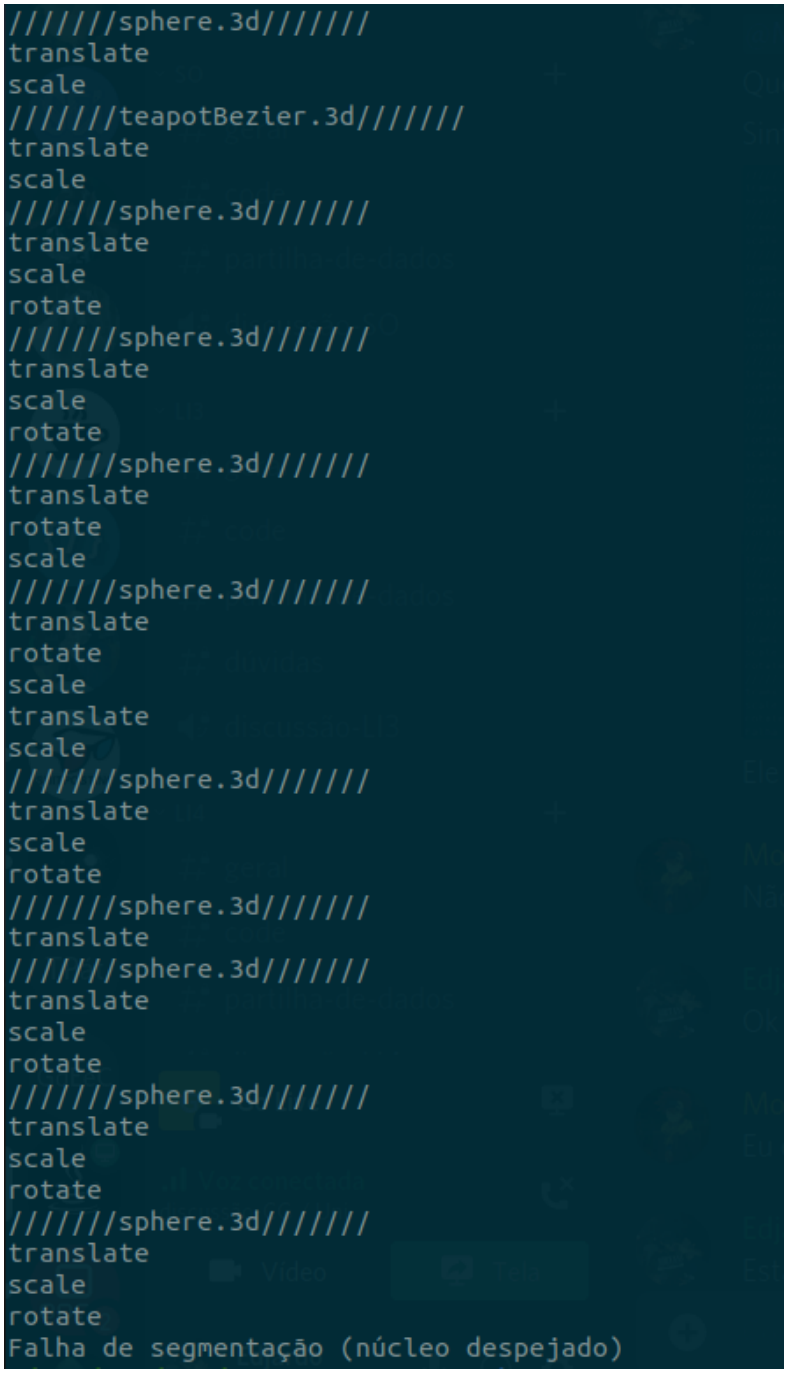

Figura 3.5: Resultados e falha obtida

Capítulo 4

Conclusão

Nesta fase aprendemos a analisar Belzier Patches e a convertê-los em modelos que a nossa engine possa ler. Toda a matemática envolvida foi abordada tanto nas aulas teóricas com nas práticas. Embora tenha sido pedido a resolução através de vbos o grupo não consegiu alcançar o sucesso nesta medida especialmente na ordem dos vbos. Esta parte do trabalho foi a que originou mais dificuldades que, por agora não conseguiram ser ultrapassadas. Posto isto, o nosso objetivo para a próxima fase, não esquecendo os objetivos dessa mesma fase, teremos em vista terminar e por a funcionar aquilo que não foi possível desta parte, e elevar a metodologia de trabalho para conseguirmos elaborar um trabalho ao nível do que o grupo considera satisfatório. Focando-nos agora em pontos mais positivos, o grupo conseguiu cumprir com a parte do generator, e julgamos que, assim que a parte dos VBOs seja corrigida, a parte das trajetórias com as curvas de Catmull Rom também funcionará da forma pretendida.