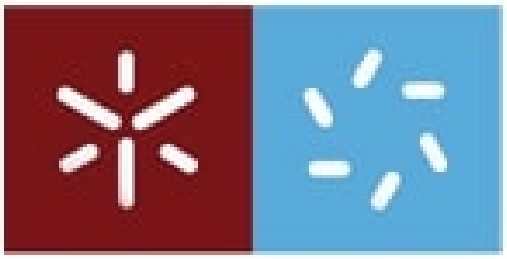

Universidade do Minho
Escola de Ciências

Computação Gráfica

Jorge Daniel Pereira Silva (a80931)
José Pedro Silva Ferreira (a96798)
Rui Alexandre Dias Neto (a80433)

26 de abril de 2024

Conteúdo

1 Introdução

Nesta terceira fase do trabalho prático foi proposto uma otimização da engine dando a possibilidade de duas novas transformações e a criação de uma nova primitiva gráfica usando as superficies e curvas de bezier.

2 Generator

Nesta fase, o **generator** deve ser capaz de criar um novo tipo de modelo baseado em patches de Bezier.Para isso criamos dentro da pasta src uma nova pasta chamada patches que vai recorrer ao ficheiro fornecido pelo professor.

2.1 Patches

Começamos por definir a matriz bezier(calculada nas aulas). Após isso desenvolvemos uma função,com o nome bezier, para calcular a formula desenvolvida nas aulas(figura 1).Após isso começamos a ler e guardar os patches e os pontos fornecidos no ficheiro *teapot.patch*.Imediatamente a seguir percorremos os patches e adicionamos o x,y,z de cada ponto à sua respetiva matriz.No final calculamos os vértices dos triângulos(figura 2) necessitando da função bezier tendo também em conta a *tessellation*

$$p(u,v) = [u^3 \quad u^2 \quad u \quad 1]M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Figura 1: Bezier Patches

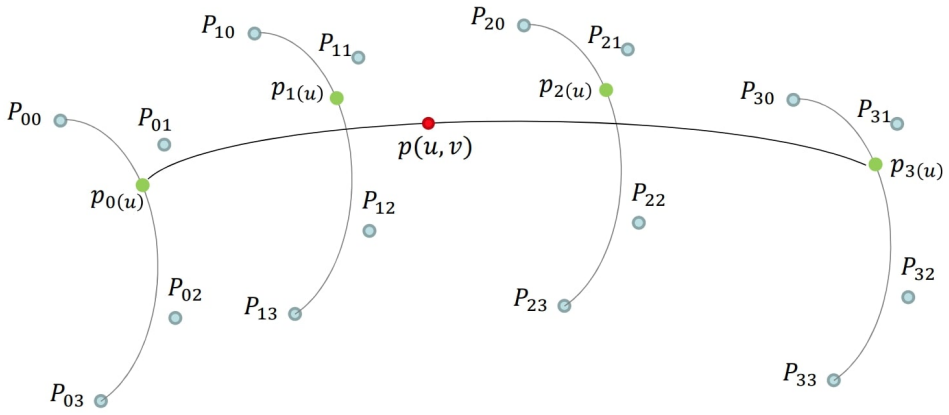

Figura 2: Ponto bezier

3 Engine

Nesta aplicação tivemos que modificar a classe *transformation* e implementar **VBOs** de modo a ter uma melhor performance.

3.1 VBOs

Para implementar **VBOs** seguimos o que aprendemos nas aulas práticas. Começamos por ir à função que lê e guarda os pontos do ficheiro,gerando e inicializamos os *buffers*(figura 3),após isso fomos à função onde desenha o *model* e demos *set* do *buffer* e utilizamos os **vbos** para desenhar os triângulos(figura 4). Implementamos também a funcionalidade de verificar o número de **FPS** no titulo da janela(figura5).

```
// VBO para os vértice
glGenBuffers(1, &(this->vertexBuffer));
glBindBuffer(GL_ARRAY_BUFFER, this->vertexBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * points.size(), points.data(),
             GL_STATIC_DRAW);

// VBO para os índices
glGenBuffers(1, &(this->indexBuffer));
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, this->indexBuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(int) * triangleIndices.size(),
             triangleIndices.data(), GL_STATIC_DRAW);
```

Figura 3: Buffer Initialization

```
// desenhar o model atraves dos vbos
void drawModel(Model model) {
  // Set buffer active and define the semantics
  glBindBuffer(GL_ARRAY_BUFFER, model.get_points());
  glVertexPointer(3, GL_FLOAT, 0, 0);
  // Draw the model using the index buffer
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, model.get_index());
  glDrawElements(GL_TRIANGLES, model.get_numPoints() * 3, GL_UNSIGNED_INT, NULL);
}
```

Figura 4: VBOs Drawing

```
// Calculate FPS
frame++;
int time = glutGet(GLUT_ELAPSED_TIME);
if (time - timebase > 1000) {
  float fps = frame * 1000.0 / (time - timebase);
  timebase = time;
  frame = 0;
  // Update window title with FPS
  char s[30];
  sprintf(s, "FPS: %4.2f", fps);
  glutSetWindowTitle(s);
}
```

Figura 5: FPS

3.2 Transformation

Um dos objetivos desta fase era estender os elementos da rotação e da translação. Em relação à translação, foi fornecido um conjunto de pontos para definir uma curva cúbica Catmull-Rom, assim como o número de segundos para percorrer toda a curva. Esta transformação também inclui um campo *align* para especificar se o objeto deve alinhar-se com a curva.Na rotação, o ângulo pode ser substituído pelo tempo, significando o número de segundos para realizar uma rotação completa de 360 graus em torno do eixo especificado. Para implementar tudo isto,redefinimos a função *parsetransformation* e também tivemos que alterar a função *drawmodels* de modo a ter atenção a estas novas transformações.

3.3 CatmullRom

Este ficheiro contém as funções necessárias para representar as curvas do *Catmull-Rom*. Reutilizamos as funções desenvolvidas nas aulas práticas. A função *buildRotMatrix* cria uma matriz de rotação dados 3 vetores x,y e z e devolve o resultado para um *array* m. A função *cross* faz a multiplicação de dois vetores. A função *normalize* normaliza um *array* que é passado como input. A função *multMatrixVector* multiplica uma matriz por um vetor. A função *getCatmullRomPoint* é responsável por computar a curva de *Catmull-Rom*. A função *getGlobalCatmullRomPoint* é responsável por obter os próximos pontos da curva para um dado valor t. A função *renderCatmullRomPoint* é a função que vai desenhar as curvas. A função *drawCatmullRomCurve* é responsável por fazer a translação.

3.4 Resultados

Demonstramos aqui os ficheiros XML lidos e o seu resultado:

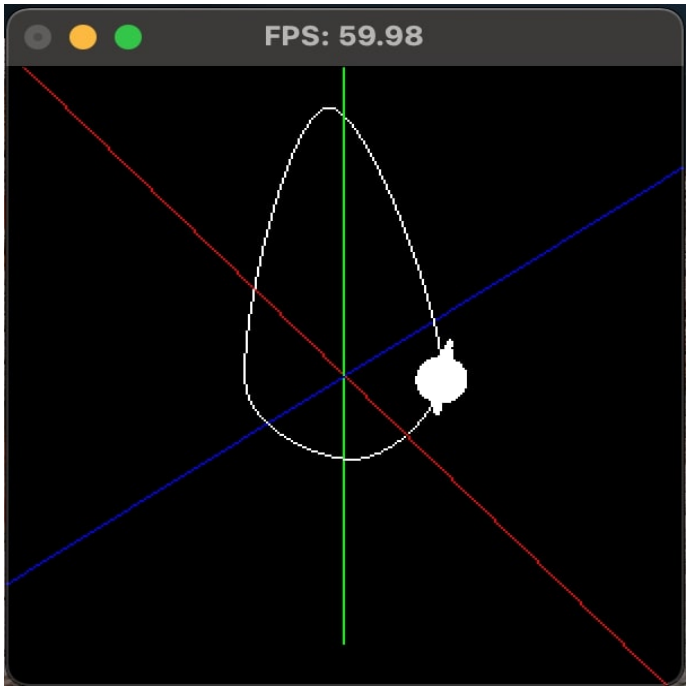

Figura 6: Teste1

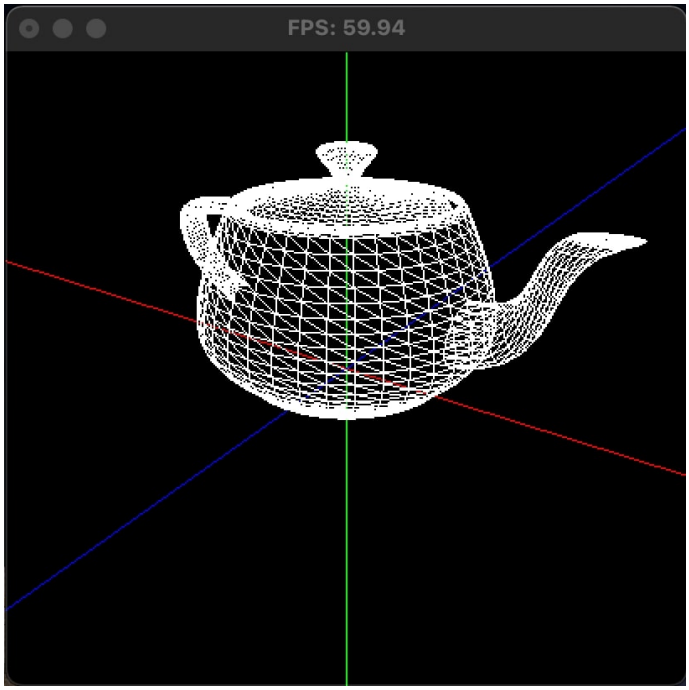

Figura 7: Teste2

4 Conclusão

Durante a realização desta terceira fase tivemos que lidar com situações difíceis como criar a nova primitiva gráfica usando um conceito completamente novo. Também introduzimos o conceito das curvas de *Catmull-Rom* que contribui para desenhar as órbitas dos planetas e das suas respetivas luas. Concluindo,com a ajuda das aulas teóricas, aulas práticas e slides conseguimos realizar o pretendido para esta terceira fase.