

Computação Gráfica

Trabalho II

Labirinto Com Curvas

INTRODUÇÃO

Este exercício consiste em desenvolver, em OpenGL, um jogo em que um veículo se desloca ao longo de ruas definidas por curvas paramétricas do tipo Bèzier.

O veículo pode andar para frente, parar, ou andar para trás.

Nestas ruas, também se deslocam, pelo menos, 10 veículos inimigos. Se houver colisão entre os inimigos e o veículo do jogador, o jogo termina.

No link a seguir se pode ver uma parte da implementação do trabalho:

<https://youtu.be/58N0vPufk4s>

Para o desenvolvimento do trabalho, recomenda-se a utilização do código-base que está no Moodle, no mesmo card onde está esta definição. Este código gera a tela a seguir

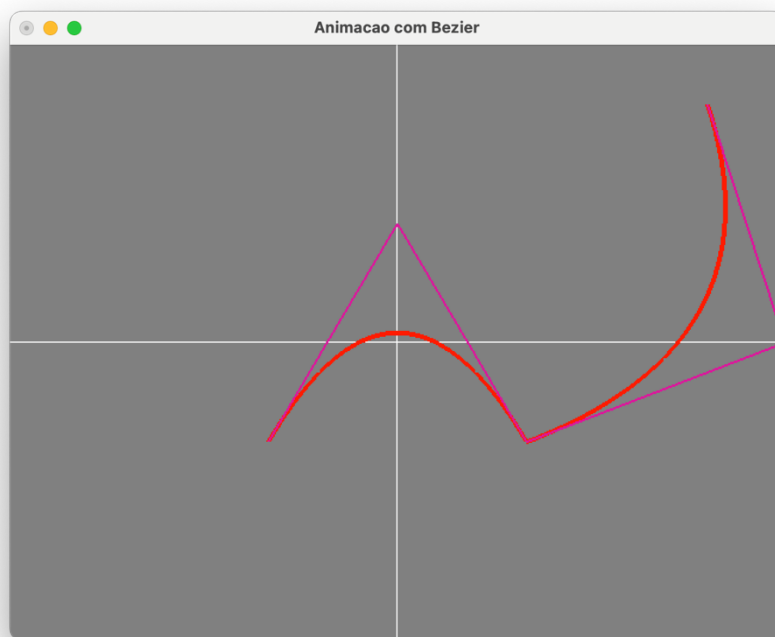


Figura – Programa-Base

RUAS

As ruas por onde os veículos podem trafegar devem ser formadas por uma sequência de curvas paramétricas do tipo Bèzier de 3 pontos.

Devem existir pelo menos 20 curvas.

As coordenadas dos pontos de controle das curvas devem ser definidas em um arquivo a ser lido com a classe Polígono, disponível no código-base do Trabalho.

A sequência de pontos que formam cada uma das curvas deve ser definida como índices que referenciam a lista de pontos. No exemplo a seguir, são definidas 13 curvas, a partir de 7 pontos de controle.

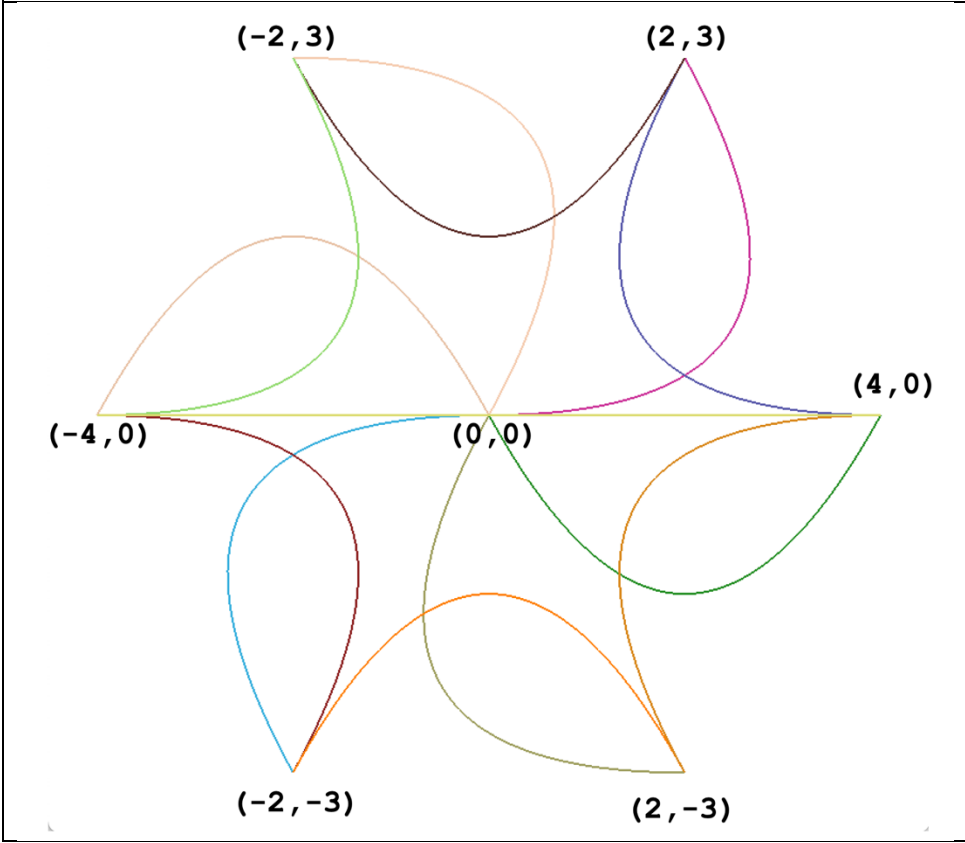
Imagem as Curvas	Coordenadas dos pontos de controle	Curvas
	7 0 0 4 0 2 3 -2 3 -4 0 -2 -3 2 -3	13 0 1 2 0 2 3 0 3 4 0 4 5 0 5 6 0 6 1 1 0 2 2 0 3 3 0 4 4 0 5 5 0 6 6 0 1 4 0 1

Figura – Exemplo de curvas

VEÍCULO DO JOGADOR

O veículo do jogador deve ter uma forma tal que seja diferente na parte da frente, em relação à parte de trás. O objetivo é permitir que se visualize qual é a dianteira e qual a traseira do veículo.

O veículo deve deslocar-se sempre alinhado com a trajetória, como nos exemplos a seguir.



Figura – Orientação do veículo de acordo com a trajetória

O sentido do deslocamento deve ser definido pelo pressionamento de uma tecla, a qualquer momento do jogo, repetidamente.

Para parar e reiniciar o movimento, deve ser usada a tecla **ESPAÇO**.

TROCA DA CURVA

Ao chegar ao ponto final/inicial de uma curva, os personagens devem iniciar o movimento sobre uma das curvas que inicia, ou termina naquele ponto.

A escolha da próxima curva deve ser realizada de forma aleatória, quando o veículo chegar no meio da trajetória (parâmetro $t == 0.5$).

No caso do veículo do jogador, ao chegar neste ponto, a próxima curva deve ser desenhada de uma cor e espessuras diferentes.

Caso o usuário deseje, pode usar uma tecla para selecionar outra curva, de maneira circular, enquanto o veículo está se deslocando.

VEÍCULOS DOS INIMIGOS

Devem existir pelo menos 10 veículos inimigos. A forma destes veículos deve seguir a mesma regra do veículo do jogador, mas o modelo deve ser diferente e cada um deles deve ter uma cor específica.

Estes veículos devem iniciar posicionados no meio de curvas escolhidas aleatoriamente.

Para metade dos veículos inimigos, o sentido inicial do movimento deve ser em direção ao ponto inicial da curva. Para a outra metade, deve ser em sentido contrário.

SOBRE A RELAÇÃO ENTRE AS CURVAS E OS PERSONAGENS

Como podemos ter mais de um personagem sobre uma mesma curva, a estrutura de personagem, representada por uma **instância**, deve guardar uma referência para a curva onde este se encontra, bem como o valor atual do parâmetro "T".

Uma sugestão para a estrutura de **instância** é apresentada a seguir.

```
class InstanciaBZ{
public:
    InstanciaBZ();
    InstanciaBZ(Bezier *Curva); // Cria uma instancia e associa uma curva a ela

    TipoFuncao *modelo; // Referencia para a funcao que desenha o modelo

    Bezier *Curva; // referencia para a curva onde esta' a instancia

    Ponto Posicao, Escala;
    float Rotacao;

    int nroDaCurva; // Nro da curva onde esta' o personagem
    int proxCurva; // Nro da curva para onde ira' o personagem

    int cor;
    float Velocidade;

    float tAtual; // Valor do t onde esta' o personagem
    int direcao; // Andando do fim para o inicio, ou ao contrario

    void desenha();
    void AtualizaPosicao(float tempoDecorrido);
    Ponto ObtemPosicao();
};
```

SOBRE A QUANTIDADE DE PERSONAGENS

O programa seja genérico quanto ao número de inimigos.

Isto significa que **não deve ser feito** algo como o código do exemplo a seguir em que cada perso

```
Inimigo1.posiciona(x1, y1);
Inimigo2.posiciona(x2, y2);
Inimigo3.posiciona(x3, y3);
...
Inimigo10.posiciona(x10, y10);
```

Neste caso, o correto é algo assim:

```
void DesenhaPersonagens(float tempoDecorrido)
{
    for(int i=0; i<nInstancias;i++)
```

```

{
    Personagens[i].AtualizaPosicao(tempoDecorrido);
    Personagens[i].desenha();
}
}

```

SOBRE AS TRANSFORMAÇÕES GEOMÉTRICAS DO PERSONAGENS

Conforme o exemplo acima, o posicionamento e a orientação dos objetos devem seguir a ideia de **instanciamento**, ou seja, cada inimigo deve ser definido como um conjunto de parâmetros e uma referência ao modelo geométrico que é exibido. Estes parâmetros devem ser internos a classe que define o inimigo. Não deve haver cópias dos modelos, apenas referências. Além disto, as transformações devem ser realizadas a partir do uso das funções de OpenGL, e não manualmente.

SOBRE A VELOCIDADE DOS PERSONAGENS

O programa precisa ter **controle da velocidade** do movimento, ou seja, a velocidade de deslocamento de todos dos personagens é sempre a mesma. Com isto, em curvas mais longas o personagem leva mais tempo para ir do início ao fim, do que em curvas mais curtas. Isto vai requer que vocês controlem o valor do “T” durante o deslocamento sobre as curvas. O que não pode ser feito é fazer sobre algo como

```
T = T + 0.001
```

Para solucionar este problema, imaginem que a velocidade seja 4m/s. Assumindo uma curva de 40 metros, um personagem levaria 10 segundos para percorrer a curva de um lado a outro. Para tanto, deve-se calcular qual foi o deslocamento do personagem entre o frame atual e o anterior, com base em sua velocidade:

```
deslocamento = Personagens[i].velocidade*tempo;
```

Com base no deslocamento, pode-se determinar quanto o valor do parâmetro T deve mudar

```
deltaT = deslocamento/comprimentoDaCurva
T = T + deltaT
```

Para calcular o comprimento de uma curva Bèzier, pode-se usar um código como o que segue. Este código já está na classe `Bezier` disponibilizada junto com a definição do trabalho.

```

void Bezier::calculaComprimentoDaCurva()
{
    double DeltaT = 1.0/50;
    double t=DeltaT;
    Ponto P1, P2;

    ComprimentoTotalDaCurva = 0;

    P1 = Calcula(0.0);
    while(t<1.0)
    {
        P2 = Calcula(t);
        ComprimentoTotalDaCurva += calculaDistancia(P1,P2);
        P1 = P2;
        t += DeltaT;
    }
    P2 = Calcula(1.0); // faz o fechamento da curva
    ComprimentoTotalDaCurva += calculaDistancia(P1,P2);
    cout << "ComprimentoTotalDaCurva: " << ComprimentoTotalDaCurva << endl;
}

```

Entrega

Data de entrega no *Moodle* e apresentação: **26/09/2024** até o horário de início da aula.

Além da entrega do código o trabalho deverá ser apresentado em aula na mesma data. Esta apresentação deve ter no máximo 5 minutos e será feita a partir de um computador conectado ao Zoom, com a tela compartilhada com o professor e projetada para toda a turma.

Para a entrega deverá ser criado um vídeo, de até 2 minutos, mostrando o funcionamento do programa e um relatório demonstrando que o trabalho atende aos requisitos da especificação e contendo o link para o vídeo.

Durante a apresentação será avaliado o domínio da resolução do problema, podendo inclusive ser possível invalidar o trabalho quando constatada a falta de conhecimento sobre o código implementado.

Os trabalhos podem ser desenvolvidos em duplas. Os arquivos, contendo os programas-fontes do programa, devem ser compactados e submetidos pelo *Moodle* até a data e hora especificadas. **ENVIE APENAS ARQUIVOS .ZIP, ou seja, não envie 7z, rar, tar.gz, tgz, tar.bz2, etc.**

A cópia parcial ou completa do trabalho terá como consequência a atribuição de nota ZERO ao trabalho dos alunos envolvidos.

FIM