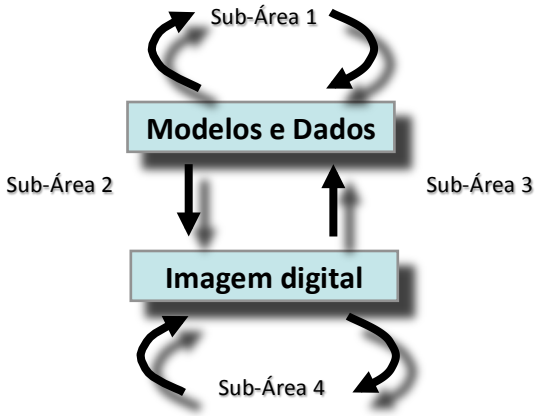
Instituto de Computação
Bacharelado em Ciência da Computação
Disciplina: Computação Gráfica
Primeira lista de exercícios - 2012.2

Conceitos fundamentais

1) A Computação Gráfica é dividida em diversas sub-áreas. O diagrama abaixo mostra uma possível classificação. Diga qual é o nome de cada uma das sub-áreas e descreva sucintamente as características das mesmas.



Sub-Área 1: Modelagem – A modelagem, tipicamente denominada modelagem geométrica, lida com problemas que envolvem a representação, geração e manipulação de formas como curvas, superfícies e sólidos, em sistemas computacionais. Problemas normalmente estudados envolvem representação de formas por subdivisão, representações em multiresolução, simplificação de malhas, modelagem a partir de imagens e etc. As técnicas provenientes da área de modelagem têm inúmeras aplicações na indústria, em problemas de física e matemática, engenharias, projeto e manufatura auxiliados por computador (CAD e CAM, respectivamente) e etc.

Sub-Área 2: Processamento de imagens – É a sub-área responsável pelo estudar técnicas para representar, manipular e realizar operações sobre imagens digitais. Ao processar uma imagem digital as técnicas de processamento de imagens produzem uma outra imagem, onde determinadas características são realçadas ou modificadas, de forma a facilitar a realização de diferentes processos que utilizam as informações

codificadas nas imagens. Cita-se, como exemplo, a aplicação de filtros para remover ou minimizar ruídos em uma imagem digitalizada por um scanner. A sub-área de processamento de imagens considera imagens, em muitos casos, como um tipo particular de sinal, havendo desta forma uma considerável interseção com a área de processamento de sinais. Problemas comumente tratados pela área são os problemas de realce de características de imagens, segmentação, transformações geométricas aplicadas a imagens, composição, além de técnicas para armazenamento e transmissão. A área de processamento de imagens possui inúmeras aplicações, como nos próprios processos de síntese de imagens, na ciência dos materiais, astronomia, geografia, microscopia, aerofotogrametria e etc.

Sub-Área 3: Analise de imagem – Trata da aquisição de informação a partir de uma imagem digital, aquisição esta muita das vezes baseada em reconhecimento de padrões e nas características dos sistemas de formação de imagens. Temos como exemplos de aplicações a identificação de placas de automóveis, a identificação de áreas desmatadas, detecção de tumores em dados médicos, calibração automática de câmeras, determinação da estrutura tridimensional de objetos a partir de imagens e outras.

Sub-Área 4: Síntese de imagens – Trata da geração de imagens a partir de um conjunto de dados e modelos. Os principais problemas estudados estão relacionados à produção de imagens realistas e visualização de dados, fenômenos e processos, em muitos casos de forma interativa e, até mesmo, em tempo real. A síntese de imagens se propõe a investigar diversos métodos, algoritmos e esquemas de representação e manipulação dos dados, de forma a solucionar tais problemas de modo eficiente e econômico. Nos estágios iniciais, a síntese de imagens introduziu os principais modelos e técnicas para geração de imagens 3D em dispositivos raster, tais como, estruturas de dados para representação de objetos gráficos 2D e 3D, projeções e modelos de câmera, algoritmos para rasterização de polígonos e recorte, remoção de superfícies escondidas e iluminação direta, técnicas para interação e geração de curvas e superfícies. Em uma fase posterior, algoritmos de iluminação mais sofisticados como Raytracing e Radiosidade e técnicas de mapeamento de textura foram propostos. Atualmente são investigadas técnicas capazes de gerar imagens cada vez mais realistas, utilizando modelos sofisticados com o auxílio do avanço tecnológico das placas e processadores gráficos, assim como dos dispositivos de captura e visualização. A Síntese de Imagens tem grande aplicação na indústria, nas

engenharias, nas diversas áreas da ciência, arquitetura, indústria do entretenimento, medicina e etc.

2) Descreva alguns dos principais dispositivos de entrada e de saída utilizados em Computação Gráfica.

Os dispositivos de entrada e saída são responsáveis pela interação entre o usuário, no sentido amplo, e a maquina. Os dispositivos de entrada são os de captação de informações gráficas e os dispositivos de saída são os responsáveis pela visualização dos dados.

Tratando-se de dispositivos de entrada e saída gráficos, estes estão relacionados com o formato dos dados com que trabalham. Dois são estes formatos: raster (*ou matricial*) *e vetorial.*

Os dispositivos do tipo raster ou matriciais, representam os dados a partir de uma matriz MxNxC. Esta matriz é vista de forma tri-dimensional onde M representa o número de colunas, N o número de linhas e C representa a cor. Cada posição da matriz é um pixel da imagem e seu conteúdo é representado por C. Exemplos de dispositivos matriciais de entrada são: scanners e a máquinas fotográficas digitais. Como exemplos de dispositivos matriciais de saída temos o monitor CRT ou LCD e as impressoras.

O segundo formato representa a imagem de forma vetorial; as informações são armazenadas na forma de coordenadas de um espaço vetorial. Exemplos de dispositivos de entrada vetoriais são: light pen, tablet, touch pannel, 3D-digitizer e os mais comuns e conhecidos de todos: o mouse e o joystick, os quais possuem um sistema de coordenadas absolutas. Os dispositivos de saída vetorial são os que produzem imagens traçando segmentos de retas e curvas descritas por coordenadas de seus pontos iniciais e finais. Exemplos desses dispositivos são: display caligráfico, display de armazenamento e os traçadores.

3) Defina o conceito de Objeto Gráfico. Como os objetos gráficos podem ser categorizados? Utilize exemplos para descrever os diferentes tipos de objetos gráficos.

O conceito matemático diz que um objeto gráfico é um subconjunto $S \subset R^m$ *associado a uma função de atributos* $f:R^n \rightarrow R^m$. *O subconjunto S define o suporte geométrico do objeto gráfico O.*

Objetos gráficos são classificados em objetos gráficos planares ou espaciais. A dimensão de um objeto gráfico é dada pela dimensão do suporte geométrico. Curvas são objetos unidimensionais, regiões e superfícies são exemplos de objetos bidimensionais, enquanto que volumes e sólidos são exemplos de objetos tridimensionais.

Objetos gráficos podem ser também categorizados em objetos planares ou espaciais. Objetos planares são aqueles que moram em um espaço bidimensional. Exemplos de objetos planares são as regiões e curvas no plano. Objetos espaciais são aqueles em que o espaço ambiente, onde eles estão imersos, possuem dimensão maior ou igual a 3. Exemplos são as curvas no espaço, superfícies e sólidos.

4) Dê exemplos de funções de atributos de objetos gráficos bidimensionais.

Curvas

5) A descrição paramétrica de uma curva planar é definida por uma função $\gamma : I \subset R \rightarrow R^2$ tal que $\gamma(t)=(x(t),y(t))$. Explique com suas próprias palavras o que é uma curva paramétrica e ilustre com um exemplo.

Uma curva paramétrica planar é determinada através do mapeamento de um conjunto de valores em um intervalo na reta, descrito pelo parâmetro t da curva, em um par de coordenadas do plano x(t) e y(t), onde x e y são funções de t.

Uma curva paramétrica planar pode ser vista como a trajetória de um ponto que se desloca no plano, se interpretarmos o parâmetro t como o tempo. O conjunto de pontos de uma equação paramétrica planar $\gamma(t)$ descreve o que chamamos de traço da curva. Existem várias parametrizações possíveis para uma curva. Abaixo temos, como exemplos, o círculo e a espiral em representação paramétrica.

Círculo: $(\cos(t),\text{sen}(t))$, $0 \leq t \leq 2\pi$

Espiral: $(at\cos(t), at\,\text{sen}(t))$, $0 \leq t \leq 2k\pi$, $a,k \in R$

6) Defina o conceito de uma curva poligonal. Descreva as principais vantagens e desvantagens de sua utilização.

7) Descreva um método para gerar uma aproximação poligonal de uma elipse. Suponha que a elipse esteja representada através da equação paramétrica. (1.0 ponto):

$$f(u) = \begin{cases} x(u) = 2\ \cos(u) \\ y(u) = 3\ sen(u) \end{cases}$$
$$0 \le u \le 2\pi$$

Solução:

*Determinar um conjunto finito formado por **n+1** amostras da curva através da aplicação da função paramétrica sobre **n+1** pontos, no intervalo de parâmetros, determinados através de uma partição uniforme. O **n+1**-ésimo ponto tem as mesmas coordenadas do primeiro ponto (na posição 0) da seqüência, de forma a gerar uma curva fechada. Em seguida, desenhar cada segmento conectando amostras consecutivas segundo a ordem determinada pela ordem na partição uniforme.*

Algoritmo

$\Delta u = 2\pi / n$

Para i=0 ate n faça

*Poligonal[i].x = 2*sen(i*Δu)*

*Poligonal[i].y = 3*sen(i*Δu)*

Fim-para

Para i=0 até n-1 faça

DesenhaLinha(Poligonal[i],Poligonal[i+1])

Fim-Para

Fim-Algoritmo

8) Descreva um algoritmo capaz de verificar se uma curva poligonal possui auto-interseção e indique sua complexidade.

Um algoritmo força bruta com complexidade O(n2), onde n é o número de segmentos da curva poligonal, pode ser obtido através do seguinte conjunto de passos:

```
s – array com os segmentos da curva
Função DetectarSimplicidade(s):lógico
Para i:←1 até n faça
        Para j:←i+1 até n faça
                Se Intersectar(s[i],s[j]) então
                        Retornar verdadeiro
                Fim_Se
        Fim_Para
Fim_para
Retornar falso;
```

Um algoritmo mais eficiente, como complexidade O(n log n), onde n é o número de segmentos da curva poligonal, pode ser obtido através da ordenação lexicográfica (ordenação considerando as coordenadas x e y dos pontos) extremos do segmento, seguido da técnica de varredura do plano, a qual utiliza uma estrutura de dados L, do tipo dicionário, que armazena o status de linhas de varredura e uma estrutura que armazena eventos E, que no caso é apenas um array que armazena os pontos extremos dos segmentos.

A estrutura L armazena os segmentos conforme a relação $>_x$. Dados dois segmentos s_1 e s_2, comparáveis em uma abscissa x, isto é, se existe uma linha vertical passando pela abscissa x que intersecta tanto s_1 e s_2, dizemos que $s_1 >_x s_2$ se a interseção de s_1 com x for maior que a interseção de s_2 com x.

A relação $>_x$ define uma ordem total e somente muda nos seguintes casos:

- *O extremo esquerdo de um segmento s é encontrado e nesse caso s deve ser inserido em L.*

- *O extremo direito de um segmento s é encontrado e nesse caso s deve ser removido de L, por não mais ser comparável com os demais segmentos.*
- *Um ponto de interseção entre dois segmentos s1 e s2 é alcançado e nesse caso s1 e s2 mudam de ordem na ordenação.*

A estrutura de dados L suporta as operações:

Inserir(L,s) – insere em tempo logaritmo um segmento s.

Deletar (L,s) – remove em tempo logaritmo um segmento s.

Acima(s, L) – retorna em tempo constante o segmento acima de s em L.

Abaixo(s,L) – retorna em tempo constante o segmento abaixo de s em L.

O algoritmo apresentado abaixo, devido a Bentley e Ottmann, considera apenas o problema de determinar a existência de uma interseção ou não, isto é um problema de detecção.

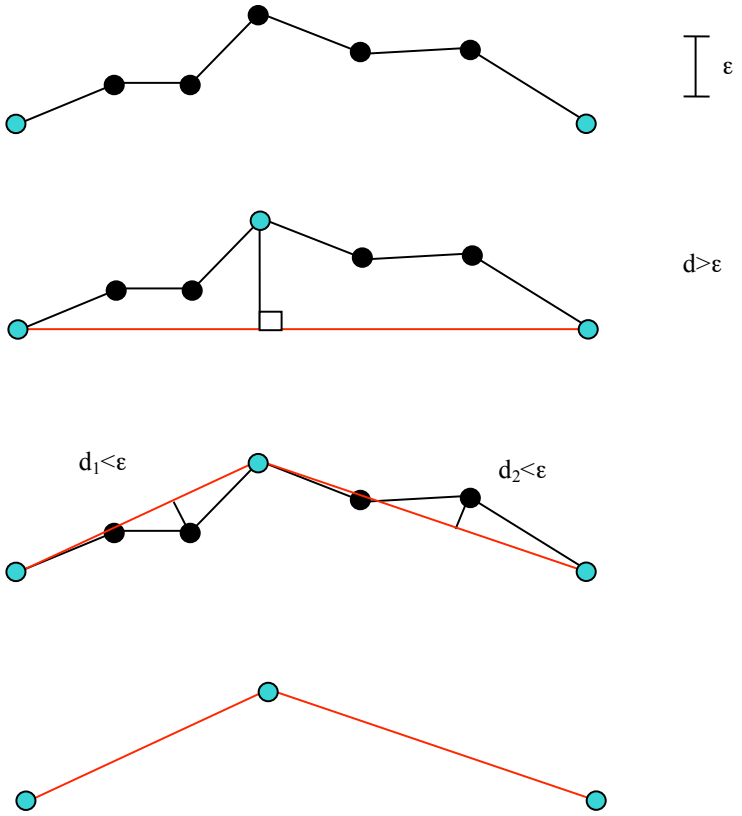
```
s – array com os n segmentos da curva pontos – array com os m pontos dos segmentos da linha poligonal
Função DetectarSimplicidadeVarreduraDoPlano(s):lógico
pontos – array com os m pontos dos segmentos da linha poligonal
OrdenarLexicograficamente(s,pontos) { ordenar os pontos dos segmentos em função de x e y e armazenar no array pontos}
Para i← 1 até m faça;
        p ←pontos[i];
        s ← segmento em que p é um ponto extremo;
        Se (p é um ponto extremo esquerdo) então
                Inserir(L,s)
                s1←Acima(L,s)
                s2←Abaixo(L,s)
                Se Intersectar(s1,s) então
                        Retornar verdadeiro
                Fim_Se
                Se Intersectar(s2,s) então
                Retornar verdadeiro
                Fim_Se
        Senão {p é um ponto extremo direito}
                s1←Acima(L,s);
                s2←Abaixo(L,s);
                Deletar(L,s);
                Se Intersectar(s1,s2) então
                        Retornar verdadeiro
                Fim_Se
        Fim_Se
Fim_Para
Retornar falso;
```

O algoritmo somente retorna uma interseção se ela é encontrada e nunca retorna uma interseção inexistente. Também não deixa de apresentar pelo menos uma interseção quando interseções existem. Para isto basta mostrar que ele sempre retorna a interseção mais a esquerda qL (a primeira da esquerda para direita). Suponha que qL coincida com um ponto extremo esquerdo p de um segmento, então, neste caso, tal interseção é encontrada no processamento de p. Suponha agora o caso em que qL não seja um ponto extremo esquerdo p de um segmento. Neste caso, a estrutura L mantém a ordenação >x, correta a esquerda de qL. Além disso, existe um pequeno intervalo de abscissas à esquerda de qL em que dois segmentos que se intersectam em qL encontram-se adjacente em L, o que faz com que qL acabe sendo descoberta em algum passo do algoritmo.

Como as seleções requerem tempo logarítmico O(log n) no pior caso, a complexidade da detecção das interseções é O(m log n) que é igual a O(n log n), já que o número de pontos m da curva poligonal é 2n-1, onde n é o número de segmentos. Tal custo coincide com o custo da ordenação o que indica que a complexidade total é O(n log n). Observe que somente O(n) testes de interseção são realizados o que pode fazer com que algumas interseções não sejam detectadas, entretanto isto não importa, já que o algoritmo sempre detecta a existência de pelo menos uma interseção, caso ela exista.

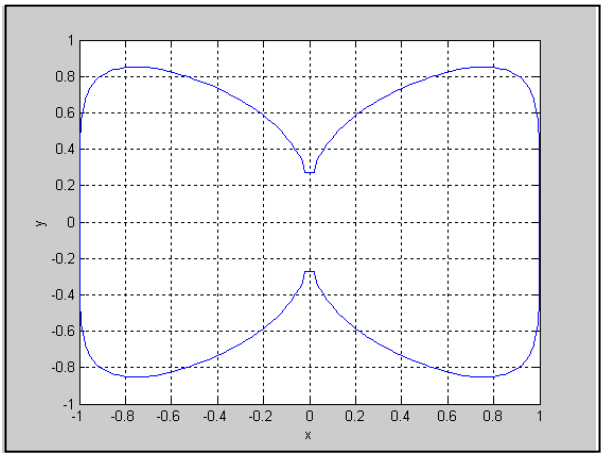
9) Faça uma pesquisa sobre o algoritmo Douglas-Peucker para simplificação de curvas poligonais. Descreva algumas de suas aplicações (1.0 ponto).

O algoritmo de Douglas-Peucker é um algoritmo de simplificação de curvas poligonais bastante simples que utiliza um processo de inserção sucessiva de pontos de acordo com uma medida de erro especificada. A cada passo, o algoritmo tenta aproximar uma sequência de pontos por um segmento de reta ligando o primeiro ao último ponto. Uma vez encontrado o ponto mais distante entre a curva e o segmento aproximador, verifica-se se sua distância está dentro de um limite estabelecido. Se a resposta for positiva a aproximação é aceita, caso contrário o algoritmo é aplicado recursivamente às subsequências anterior e posterior ao ponto mais distante.



10) Explique o que é uma curva implícita e como é possível determinar a posição de um ponto em relação a uma região descrita por uma curva implícita fechada (testar se o ponto é interior, exterior ou se está sobre a curva que delimita a região). Dê um exemplo para ilustrar sua explicação.

11) Considere a curva dada pela equação $f(x,y) = x^6+y^6-x^2$. Classifique cada ponto da lista L = {(0.4,0.6),(0.0,1.0),(-0.2,0.0),(0.8,0.3) } como *interior* ou *exterior*. Descreva o método utilizado para classificar os pontos.



Dados do problema:

Polinômio: $f(x,y) = x^6+y^6-x^2$

Lista de pontos:

L = { (0.4,0.6), (0.0,1.0), (-0.2,0.0), }

Solução:

Aplicar os pontos na equação para verificar pertence (interior) ou não pertence (exterior).

Pontos	***$f(x,y) = x^6+y^6-x^2$***	***Resultado***
(0.4, 0.6)	*$f(x,y) = 0.4^6+0.6^6-0.4^2$*	*0.210752*
(0.0, 1.0)	*$f(x,y) = 0.0^6+1.0^6-0.0^2$*	*1.0*
(-0.2, 0.0)	*$f(x,y) = -0.2^6+0.0^6-(-0.2)^2$*	*0.040064*
(0.8, 0.3)	*$f(x,y) = 0.8^6+0.3^6-0.8^2$*	*0.902873*

Baseando-se na tabela e no teorema de Jordan, todos os pontos pertencem ao plano.

Regiões

12) Considere uma aplicação onde o usuário tem que selecionar uma dada região em um mapa exibido na tela, clicando com um botão do mouse. Descreva como o problema de se determinar a região selecionada pode ser resolvido conhecendo-se as coordenadas do ponto clicado? Como as regiões devem ser representadas para a estratégia funcionar?

13) Implemente um método para poligonização de curvas representadas de forma implícita (ver página do curso).

14) Descreva um método para determinar se um segmento intersecta ou é interior a um polígono.

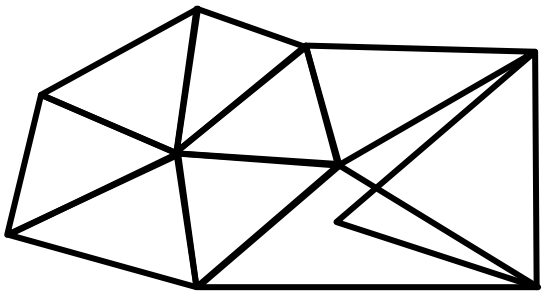
Triângulações

15) Uma triangulação de uma região do plano é definida como uma coleção T = {Ti} de triângulos tal que, para dois triângulos distintos Ti e Tj em T com $T_i \cap T_j \neq \emptyset$, temos:

- $T_i \cap T_j$ é um vértice em comum ou,

- $T_i \cap T_j$ é uma aresta em comum.

Sabendo-se disto, justifique porque a triangulação abaixo está incorreta (2.0 pontos):



A triangulação está incorreta porque existe um triângulo que sobrepõem outros dois na subdivisão do plano, o que fere a regra que define o que é uma triangulação em termos das interseções válidas.

16) Faça uma pesquisa sobre o método de triangulação Ear Clipping (1.0 ponto).

O algoritmo Ear Clipping é uma técnica para triangularizar uma região planar que se baseia na remoção sucessiva de "orelhas" de um polígono. Uma orelha é triângulo T formado por três vértices sucessivos V_0, V_1 e V_2 para os quais não existe nenhum outro vértice do polígono que seja interior a T. O vértice V_1 é considerado a ponta da orelha e a linha passando por V_0 e V_2 é a diagonal do polígono. Existe um teorema que afirma que um polígono com quatro ou mais lados sempre tem pelo menos duas orelhas, o que sugere um algoritmo recursivo para triangulação (ver detalhes na referência abaixo). Basta então localizar uma orelha em um polígono com $n \geq 4$ vértices e removê-la resultando em um polígono com n-1 vértices, sobre o qual se repete o mecanismo até que não reste nenhum triângulo.

Referência:
http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf (acessado em 22/08/11).

Transformações Geométricas no plano

17) Considere as seguintes figuras geométricas abaixo. Determine a transformação necessária para levar a figura 1 na figura 2 (2.0 ponto).

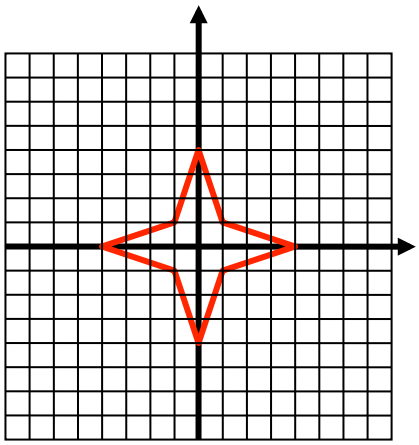


Figura 1

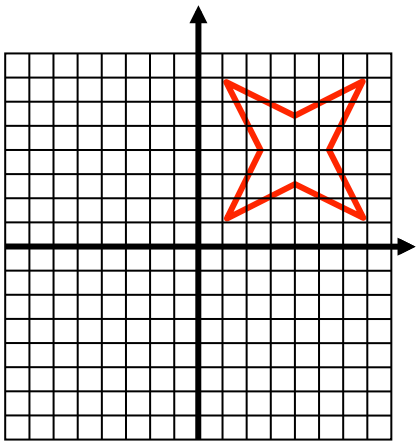


Figura 2

As operações de transformação necessárias para levar a figura 1 na figura 2 requerem primeiramente a rotação da figura em torno da origem de um ângulo $\alpha = 45^\circ$ *no sentido anti-horário, seguida de uma translação dada pelo vetor de translação (4,4). Em forma matricial as matrizes de translação* $Matriz_A$ *e* $Matriz_B$*, em coordenadas homogêneas, conforme descrito abaixo*

Dados do problema:

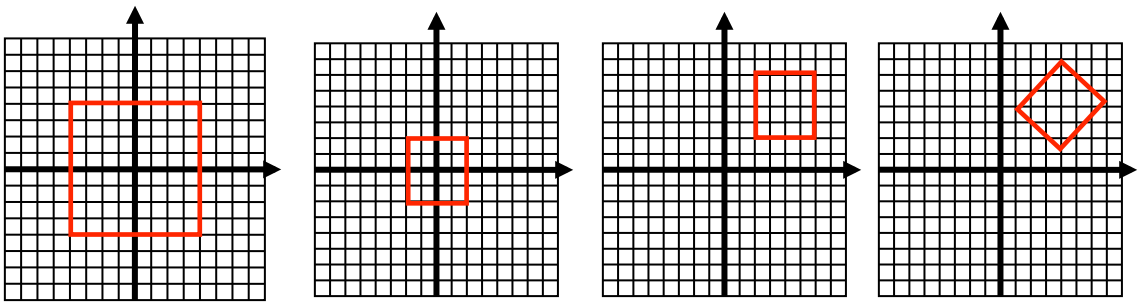
$$\boldsymbol{Matriz}_A = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \text{ e } \boldsymbol{Matriz}_B = \begin{bmatrix} \cos 45^\circ & -\boldsymbol{sen}45^\circ & 0 \\ \boldsymbol{sen}45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A $Matriz_C$ *efetua a transformação final e é dada pelo produto* $Matriz_A$ *x* $Matriz_B$ *:*

$$\boldsymbol{Matriz}_C = \begin{bmatrix} \cos 45^\circ & -\boldsymbol{sen}45^\circ & 4 \\ \boldsymbol{sen}45^\circ & \cos 45^\circ & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

A figura é então finalmente transformada efetuando-se o produto da $Matriz_C$ *pelos vetores em coordenadas homogêneas correspondentes a cada vértice da estrela.*

18) Uma matriz de transformação pode representar a composição de diferentes transformações como, por exemplo, translações, rotações e escalas. Isto permite que uma seqüência de transformações sobre um objeto possa ser representada através de uma única operação matricial aplicada a cada um de seus vértices. Observe as transformações aplicadas no quadrado abaixo:



→ Escala → Translação → Rotação

(+45°)

Escreva a matriz de transformação resultante para a seqüência de transformações aplicadas ao quadrado. (Considere que as transformações ocorrem no plano e que os pontos estão representados em coordenadas homogêneas) .

Solução:

A primeira operação corresponde a uma escala cujo fator é 0.5, tanto na direção x quanto na direção y. Isto pode ser expresso através da seguinte matriz de transformação 2D em coordenadas homogêneas:

$$\boldsymbol{A} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A segunda transformação é uma translação pelo vetor (4,4), o que é expresso pela matriz

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

A última transformação corresponde a uma rotação de π/2 graus no sentido anti-horário em torno do centro da figura que está no ponto (4,4). Para isso é necessário levar a figura na configuração corrente de volta a origem, aplicar a devida rotação e transladá-la de volta à posição (4,4). Tal combinação é expressa através da matriz:

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\pi/2) & -\boldsymbol{sen}(\pi/2) & 0 \\ \boldsymbol{sen}(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

Finalmente, a seqüência de transformações T é representada pelo produto das matrizes:

$$\boldsymbol{T} = \boldsymbol{CxBxA} = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\pi/2) & -\boldsymbol{sen}(\pi/2) & 0 \\ \boldsymbol{sen}(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Podemos observar, pelo resultado da composição que, para efeito final, a transformação dada pela matriz B é desnecessária, já que se anula com a componente de translação à direita, associada à transformação representada pela matriz C.

OpenGL

19) Escreva uma função, utilizando OpenGL, que desenhe um cubo na tela. Observação: não utilizar funções da GLUT.

20) Uma aplicação em OpenGL costuma ser organizada através de 3 partes: inicialização, loop principal e finalização. Explique o que vem a ser o loop principal e o que se costuma processar nesta etapa.

21) O OpenGL funciona com uma arquitetura baseada numa máquina de estados. Explique esta afirmação e dê 2 exemplos.

O OpenGL funciona baseado em uma máquina de estados, pois antes de serem enviados polígonos para serem processados, estipula-se os estados referentes a diversos aspectos do pipeline gráfico, seguindo-se a sintaxe:

```
glEnable(GL_ATRIBUTO_XXX);
```

Na seqüência, quando a API for realizar a visualização, irá usar o estado pré-estabelecido para diversos parâmetros.

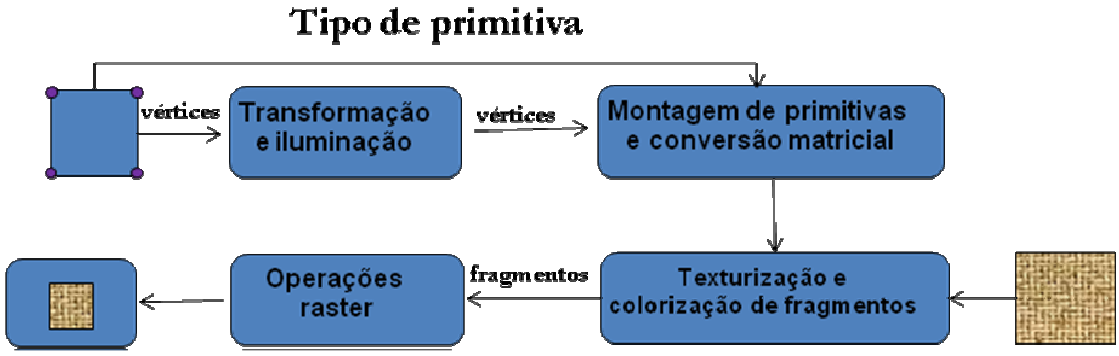
Exemplos:

```
glEnable(GL_TRIANGLE_STRIPS)

glEnable(GL_FLAT)
```

22) Explique o pipeline da OpenGL. Use um diagrama para explicar o fluxo de dados no pipeline.

Uma versão simplificada do pipeline gráfico do OpenGL, considerando a alimentação de primitivas vetoriais (formadas por vértices) pode ser delineada através do diagrama abaixo.



No primeiro estágio são aplicadas as transformações geométricas que convertem as coordenadas do objeto para coordenadas da câmera e aplica-se a projeção especificada. Isto corresponde à transformação do vértice por meio de sua multiplicação pelas matrizes modelview e projection. Neste mesmo estágio é feito o cálculo de iluminação por vértice.

No segundo estágio é montada a primitiva conforme especificada pelo usuário. O estágio anterior não dispõe de tal informação. Em seguida é feita a conversão matricial, onde são determinadas as coordenadas dos fragmentos que compõem a primitiva. Neste estágio são determinados os valores (cor, coordenada de textura, profundidade de cada fragmento através de interpolação).

No terceiro estágio as informações calculadas para cada fragmento podem ser modificadas em função do mapeamento de textura e aplicação de neblina (fog).

Finalmente as operações raster determinam se um fragmento efetivamente chegará à tela ou não, sendo aplicados testes de stencil, alpha e profundidade.

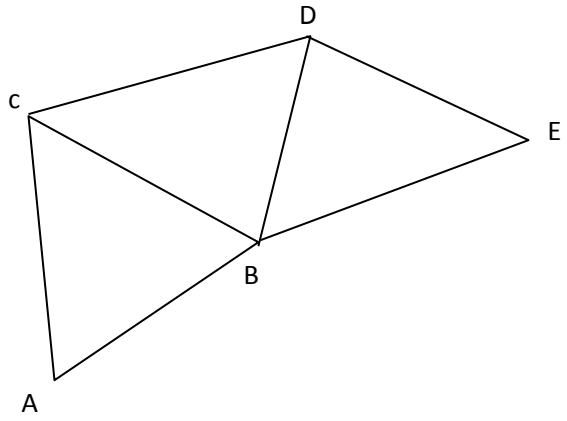
23) Escreva uma função, utilizando OpenGL, que desenhe um pentagrama na tela.

```
void Display (void)   {
  const float PI = 3.1416f;
  float angle, size = 50.0;
  int i;
  /* Limpa a tela(buffer de cores) e o buffer de profundidades */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  /* Desenha o pentagrama */
  glBegin(GL_LINE_LOOP);
  /* configura o angulo inicial em 90 graus */
  angle = PI/2.0;
  for (i = 0 ; i<=4 ; i++){
   glVertex2f(size*cos(angle), size*sin(angle));
   /* incrementa o angulo em 144 graus */
   angle += 4.0*PI/5.0;
  }
  glEnd();
  /* Troca os buffers */
   glutSwapBuffers ( );
}
```

24) O que são triangle strips e qual a sua vantagem em relação a primitiva triângulo no desenho de uma representação poligonal (2 pontos) ?

Triangles strips consistem em uma forma eficiente de se especificar uma série de triângulos conectados compartilhando vértices. Os três primeiros vértices indicados especificam o primeiro triângulo,e em seguida, cada novo vértice determina um novo triângulo formado por ele e os dois últimos vértices usados na definição do triângulo anterior na seqüência.

Tal mecanismo de representação permite uma utilização mais eficiente da memória já que ao invés de utilizar 3N vértices, onde N é o número de triângulos, utiliza apenas N+2.



Além disso, permite uma maior agilidade na transferência de dados, já que não há necessidade de transmitir informação repetida.

No exemplo abaixo, a seqüência de triângulos, formada pelos vértices A, B, C, D e E, pode ser especificada utilizando-se a primitiva GL_TRIANGLE_STRIPS simplesmente através da lista de vértices {A, B, C, D, E}. A utilização da primitiva GL_TRIANGLES requer, entretanto, o uso de seis listas {A,B,C}, {CBD}, {BED}.

25) O que são callbacks de desenho, na OpenGL?

As callbacks fazem parte fundamental do mecanismo de funcionamento da biblioteca de interface GLUT, normalmente vinculada a OpenGL.

Callbacks são funções criadas pelo programador que implementam uma função com certa assinatura, que são chamadas por um programa ou biblioteca (no caso a GLUT) quando da necessidade de se tratar algum evento. As callbacks são registradas através de funções que recebem um ponteiro para função com a assinatura especificada.

26) A qual categoria de objeto gráfico pertencem os terreno utilizados em jogos e simuladores de vôo? Explique sua afirmação.

Terrenos são tipicamente representados através da triangulação de um conjunto de vértices representando pontos sobre uma superfície que, por sua vez corresponde a um mapa de elevação. Logo, são objetos espaciais de dimensão dois (superfícies), já que não possuem área. Mais formalmente, a interseção de uma bola de raio $\varepsilon>0$ centrada em qualquer ponto que pertence a um terreno com o próprio terreno produz um disco cuja topologia é a de um pedaço do plano. Observe que tal afirmação só é válida para terrenos que podem ser expressos com uma função f(x,y). Terrenos contendo cavernas e outras estruturas que não possuem a topologia do plano não podem ser descritas por uma superfície.

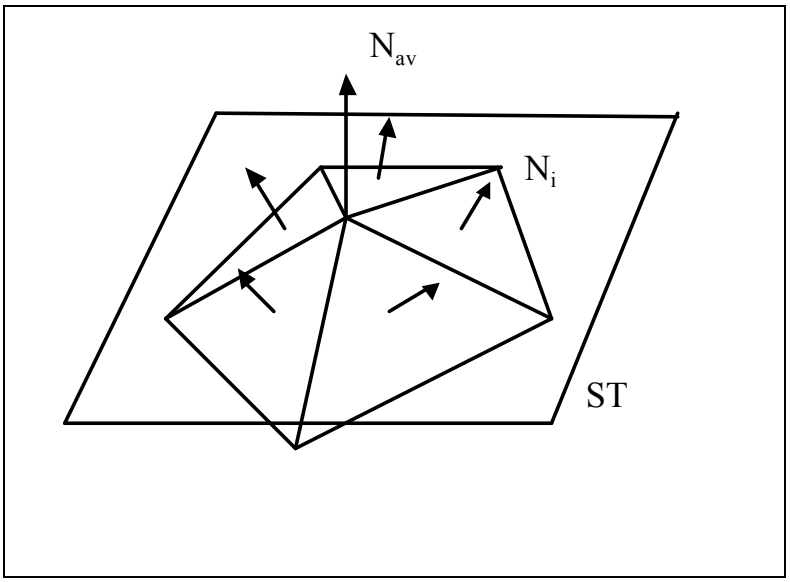
27) Descreva uma estrutura de dados capaz de representar dados de terreno.

Terrenos são normalmente descritos por triangulações que, por sua vez, são expressas através de estruturas que representam o grafo induzido pelos vértices e arestas da triangulação, juntamente com o seu grafo dual. Uma estrutura de dados bem simples pode ser obtida armazenando uma lista de triângulos onde cada triângulo faz referência aos seus vértices. Podemos tornar tal estrutura mais eficiente criando uma lista de vértices, uma lista de arestas e uma lista de faces. Cada face (triângulo) da lista de faces referencia arestas na lista de arestas e cada aresta desta última, por sua vez, referencia um par de vértices na lista de vértices. Estruturas mais sofisticadas como a winged-edge e half-edge se baseiam em construções similares. Para uso de nível de detalhe (level of detail) em terrenos podem ser utilizadas estruturas mais sofisticadas para agrupar representações do terreno em diferentes níveis de detalhe como, por exemplo, as estruturas utilizadas nos algoritmos ROAM (real-time optimally adapting meshes).

28) Um terreno pode conter muitos triângulos. Descreva uma estratégia para reduzir o número de triângulos em um modelo sem aumentar consideravelmente o erro geométrico introduzido pela simplificação. Dica: pense em como remover vértices que não são importantes da triangulação. Considere uma função dos ângulos entre a normal de um vértice v e as normais dos vértices adjacentes a v como uma medida de sua importância q(v).

Um método de dizimação(decimação) bastante elegante foi proposto no trabalho de Florian Shröder e Patrick Roßbach (Managing the complexity of digital terrain models, Comput. & Graphics,Vol. 18, No.6, pp. 775-783,1994)

para a simplificação de terrenos e sólidos geométricos em geral. O algoritmo descrito procura remover a cada passo o ponto que menos contribui para os detalhes de aproximação da superfície. O critério de seleção do ponto menos importante é baseado no grau de rugosidade em relação aos seus triângulos vizinhos. Uma superfície tangente ST é ajustada sobre um vértice p que é examinado conforme a figura abaixo:



A orientação Nav de ST é dada pela média das normais ni dos triângulos que envolvem p ponderados pela sua área Ai:

$$nav = \frac{\sum_{i=1}^{tno} \vec{n}_i . A_i}{\sum_{i=1}^{tno} A_i}$$

nav: normal média que determina a orientação da superfície tangente ST em p.

tno: número de triângulos que circundam o vértice examinado p.

$\vec{n}_i$ *: superfície normal do triângulo i.*

Ai: área do triângulo i.

Calcula-se então o ângulo máximo amax entre a normal média nav e as normais das superfícies definidas pelos triângulos que circundam o ponto p.

$$a_{\max} = \max_{i=1}^{tno} \left(\arccos \frac{\vec{n}_{av} . \vec{n}_i}{|\vec{n}_{av}| . |\vec{n}_i|} \right)$$

O resultado em amax representa o maior ângulo entre nav e ni e assume valores entre 0 e π. Se nav e todas as outras normais forem normalizadas, podemos então usar a seguinte expressão:

$$a_{\max} = \max_{i=1}^{tno} \left(\arccos \left(\vec{n}_{av} . \vec{n}_i \right) \right)$$

Pelo critério de qualidade, o algoritmo procurará remover todos os pontos cujo valor amax é menor que o valor limite estabelecido AngMax. Se o objetivo é remover certa quantidade de pontos, então deve-se ordenar previamente todos os pontos por amax em ordem crescente de forma a manter os mais significativos.

O autor deste trabalho propõe três técnicas para retriangular o conjunto de pontos resultante do processo de decimação:

***Triangulação de Delaunay**: retriangula-se segundo o critério de Delaunay todos os pontos que sobrevivem ao processo.*

***Small Hole approach**: a cada ponto removido, utiliza-se um algoritmo de triangulação simples para retriangular os "buracos" gerados.*

***Big Hole approach**: permite-se que os pontos sejam removidos e ao final do processo, retriangula-se os buracos restantes.*

29) Produza uma curva poligonal a partir da avaliação de 4 pontos uma curva de Bézier cúbica dados dados os seguintes pontos de controle (0.0,0.0), (0.3,0.8),(0.7,0.8),(1.0,0.0), (0.5 ponto).

$$x(t) = \sum_{i=0}^{3} \binom{n}{i} t^i (1-t)^{n-i} x(i)$$

$$y(t) = \sum_{i=0}^{3} \binom{n}{i} t^i (1-t)^{n-i} y(i)$$

$$x(t) = \binom{3}{0} t^0 (1-t)^3 x(0) + \binom{3}{1} t^1 (1-t)^2 x(1) + \binom{3}{2} t^2 (1-t)^1 x(2) + \binom{3}{3} t^3 (1-t)^0 x(3)$$

$$y(t) = \binom{3}{0} t^0 (1-t)^3 y(0) + \binom{3}{1} t^1 (1-t)^2 y(1) + \binom{3}{2} t^2 (1-t)^1 y(2) + \binom{3}{3} t^3 (1-t)^0 y(3)$$

$$x(t) = (1-t)^3 t^0 x(0) + 3t(1-t)^2 x(1) + 3t^2(1-t)x(2) + t^3 x(3)$$

$$y(t) = (1-t)^3 t^0 y(0) + 3t(1-t)^2 y(1) + 3t^2(1-t)y(2) + t^3 y(3)$$

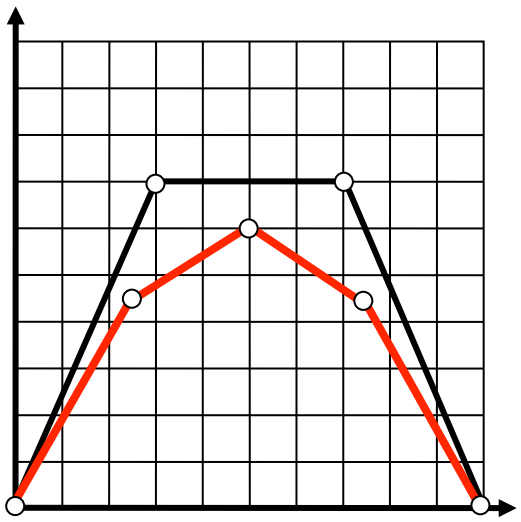
$$x(t) = (1-t)^3 t^0 0.0 + 3t(1-t)^2 0.3 + 3t^2(1-t)0.7 + t^3 1.0$$

$$y(t) = (1-t)^3 t^0 0.0 + 3t(1-t)^2 0.8 + 3t^2(1-t)0.8 + t^3 0.0$$

$$x(t) = 3t(1-t)^2 0.3 + 3t^2(1-t)0.7 + t^3 = 0.9t - 1.59t^2 + 1.69t^3$$

$$y(t) = 3t(1-t)^2 0.8 + 3t^2(1-t)0.8 = 2.4t - 2.4t^2$$

t	x(t)	y(t)
0	0.0	0.0
0.25	0.2406	0.45
0.5	0.5	0.6
0.75	0.7594	0.45
1	1.0	0.0



30) Faça uma pesquisa sobre o algoritmo de deCastlejau para geração de curvas de Bérzier (1.0 ponto).

O algoritmo de deCastlejau é um algoritmo utilizado para calcular um ponto sobre uma curva de Bérzier correspondente a um valor de parâmetro t = t0, 0≤t0≤1, baseado em aplicações repetidas de interpolações lineares. Considere, por exemplo, uma curva de Bézier de grau n=3 contendo os pontos P0,P1,P2 e P3. A parametrização do segmento P0P1 é dada por

$(1 - t)P0 + tP1,$

Para um dado valor de t=t0 podemos calcular um novo ponto de controle sobre P0P1 da seguinte forma:

$Q0 = (1 - t0)P0 + t0P1.$

Podemos aplicar o mesmo processo calculando para os segmentos P1P2 e P2P3 os pontos :

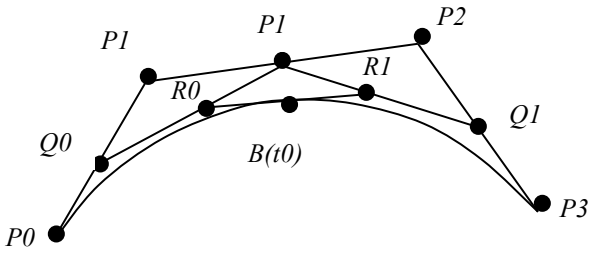
$Q1 = (1 - t0)P1 + t0P2$ e $Q2 = (1 - t0)P2 + t0P3.$

Em seguida, obtemos os pontos R0 e R1, interpolando para t=t0, respectivamente, os extremos dos segmentos Q0Q1 e Q1Q, conforme abaixo:

$R0 = (1 - t0)Q0 + t0Q1$ and $R1 = (1 - t0)Q1 + t0Q2.$

Finalmente, o ponto B(t0) na curva é obtido interpolando-se R0 e R1

$B(t0) = (1 - t0)R0 + t0R1$



31) O que é uma curva Bézier? Cite algumas propriedades.

Uma curva de Bézier é uma curva interativa descrita pela combinação (mistura) das coordenadas de pontos de controle através das funções de Bernstein. A forma geral de uma curva de Bézier de grau n é dada por

$$f(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i(u), B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

O caso mais comum de curvas de Bérzier são as curvas cúbicas onde n=3, para as quais temos as seguintes expressões para a base de Bernstein:

$$B_{i,3}(u) = \binom{3}{i} u^i (1-u)^{3-i}$$

$$B_{0,3} = (1-u)^3$$
$$B_{1,3} = 3u(1-u)^2$$
$$B_{2,3} = 3u^2(1-u)$$
$$B_{3,3} = u^3$$

Dentre as propriedadas das curvas de Bézier citamos as seguintes:

A curva se restringe ao fecho convexo uma vez que as funções de base somam 1 (um) para todo valor de u.

Os pontos de controle não exercem controle local. Mover um ponto de controle move toda a curva (As funções de base são diferentes de 0 em todo o domínio exceto em u=0 e u=1).

Os vetores tangentes à curva nos pontos extremos coincidem com a primeira e última aresta do polígono de controle.

A curva não oscila sobre nenhuma reta mais do que oscila o polígono de controle (propriedade de minimização de variação).

A curva pode ser transformada por transformações afins (translações e rotações) definidas sobre os pontos de controle.

O controle exercido pelos pontos de controle não é local. A movimentação de um ponto altera toda curva, apesar de sua influência ser maior na vizinhança de tal ponto.

Não é possível definir uma curva de Bérzier cúbica para aproximar ou representar um conjunto de n pontos sem utilizar múltiplos segmentos de curva.

O algoritmo de deCastlejau é um algoritmo utilizado para calcular um ponto sobre uma curva de Bérzier correspondente a um valor de parâmetro t = t0, $0 \leq t0 \leq 1$, baseado em aplicações repetidas de interpolações lineares. Considere, por exemplo, uma curva de Bézier de grau n=3 contendo os pontos P0,P1,P2 e P3. A parametrização do segmento P0P1 é dada por

(1 - t)P0 + tP1,

Para um dado valor de t=t0 podemos calcular um novo ponto de controle sobre P0P1 da seguinte forma:

Q0 = (1 - t0)P0 + t0P1.

Podemos aplicar o mesmo processo calculando para os segmentos P1P2 e P2P3 os pontos :

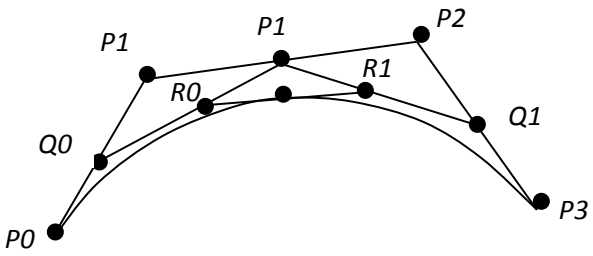
Q1 = (1 - t0)P1 + t0P2 e Q2 = (1 - t0)P2 + t0P3.

Em seguida, obtemos os pontos R0 e R1, interpolando para t=t0, respectivamente, os extremos dos segmentos Q0Q1 e Q1Q, conforme abaixo:

R0 = (1 - t0)Q0 + t0Q1 and R1 = (1 - t0)Q1 + t0Q2.

Finalmente, o ponto B(t0) na curva é obtido interpolando-se R0 e R1

B(t0) = (1 - t0)R0 + t0R1



B(t0)

32) Defina uma curva B-Spline? (1.0 ponto). O que são NURBS?

As B-Splines e as Nurbs são curvas compostas por segmentos de curva definidos com base em pontos de controle e vetores de nós. Assim como em uma B-Spline não- uniforme, nas Nurbs (Non Uniform Rational B-Spline), o espaço entre os valores dos nós não é uniforme, podendo ter multiplicidadade diferente de um.

O que de fato torna as Nurbs diferentes é a existência de um peso w_i, associado a cada ponto de controle P_i, que afeta a curva apenas localmente, assim como os pontos de controle, e que funciona como um fator de acoplamento. Quanto maior o valor de w_i, mais a curva se aproxima do ponto de controle P_i. A curva P(t) de grau k é obtida através de uma equação envolvendo uma divisão pelo somatório das bases de B-Splines ponderadas pelos pesos, o que explica o porque da curva ser considerada uma curva racional:

$$P(t) = \frac{\sum_{i=0}^{n} P_i w_i B_{i,k}(u)}{\sum_{i=0}^{n} w_i B_{i,k}(u)}$$

33) Qual a vantagem da utilização de b-splines em relação às curvas de Bérzier (1.0 ponto)?

Uma vantagem das curvas b-spline em relação a curvas de Bérzier é a de que b-splines permitem criar curvas com muitos pontos de controle, sem a necessidade de se aumentar o grau do polinômio da base ou então colar diferentes curvas de menor grau juntamente através de um mecanismo que garanta a continuidade e suavidade entre os segmentos nos pontos de junção. Isso se deve ao fato de que, por definição, b-splines descrevem curvas suaves por partes, sendo que a suavidade é garantida automaticamente através do compartilhamento de

pontos de controle entre segmentos de curva consecutivos que compõe a curva maior.

Uma outra vantagem é o grau de controle local, que se obtém como conseqüência do fato de que pontos de controle influenciam apenas um subconjunto dos segmentos que compõem a curva total, algo que não é possível de se obter através de curvas de Bézier, nas quais a modificação de um ponto de controle causa uma modificação em toda a curva.

34) Descreva um método para calcular a interseção de uma reta com uma Curva de Bézier.