Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação

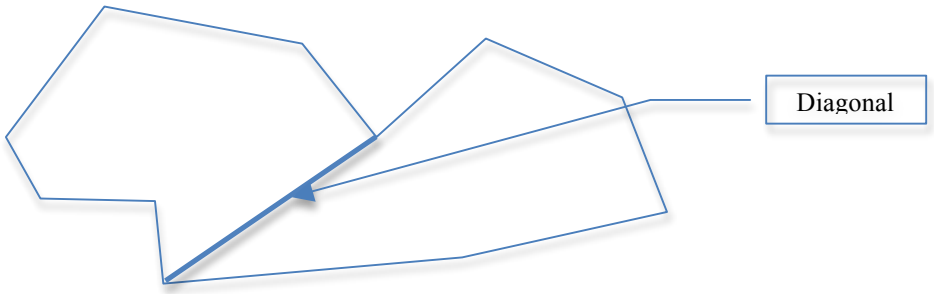
Disciplina: Computação Gráfica

AD1 - 2° semestre de 2014.

1) Em que tipos de ferramenta, usadas pela Computação Gráfica, se enquadram os aplicativos Blender e Unity 3D (1.0 ponto).

O Blender é tanto uma ferramenta de modelagem 3D quanto um Motor de Jogos (Game Engine). Já o Unity 3D é apenas um Game Engine para construção de jogos, não sendo uma ferramenta de modelagem 3D, onde um jogo pode ser criado a partir de uma composição de Game Objects, scripts e elementos importados (*assets*) para formas, texturas e audio.

2) Sabemos da Geometria Computacional que, para qualquer polígono simples *p*, com mais de três vértices, sempre é possível encontrar uma diagonal *d* em *p*. Uma diagonal de um polígono *p* é um segmento de reta completamente contido no interior de *p* que conecta dois vértices de *p*, não consecutivos.



Considerando a existência de uma função **d←EncontraDiagonal (p)**, que retorna uma diagonal *d* de um polígono *p*, apresente um algoritmo que construa uma triangulação de *p*. A triangulação é determinada por conjunto maximal de diagonais que não se intersectam. Um conjunto de diagonais que não se intersectam é dito maximal se a adição de mais uma diagonal ao conjunto provocar uma auto-interseção de diagonais (**Dica: use uma estratégia baseada em recursão. Observe que um triângulo não possui diagonais**) (1.0 ponto).

Algoritmo Triangulação (p, n): T
Entrada:
p - polígono dado por uma lista circular de vértices
n - número de lados do polígono
Saída:
T - triangulação dada por uma lista de triângulos
Início
T ← {};

Se (n =3) então
    T← p
    retorne T = p;
Fim_Se;

d ← EncontraDiagonal(p);

v1 ← d[0]; {v1 é o primeiro vértice da diagonal}
v2 ← d[1]; {v2 é o segundo vértice da diagonal}
i1 ← EncontraIndice(p,v1); {encontra a posição de v1 em p}
i2 ← EncontraIndice(p,v2); {encontra a posição de v2 em p}

{Cria as listas circulares referentes aos subpoligonos p1 e p2}
p1 ← {}
Para i = i1 ate i2 faça
    p1 ← p1 ∪ p[i];
Fim_Para
p2 ← {}
Para i=i2 ate n faça
    p2 ← p2 ∪ p[i];
Fim_Para

{Calcula recursivamente as triangulações de p1 e p2}
T1 ← Triangulação(p1,Tamanho(p1));
T2 ← Triangulação(p2,Tamanho(p2));

T = T1 ∪ T2

retorne T;
Fim

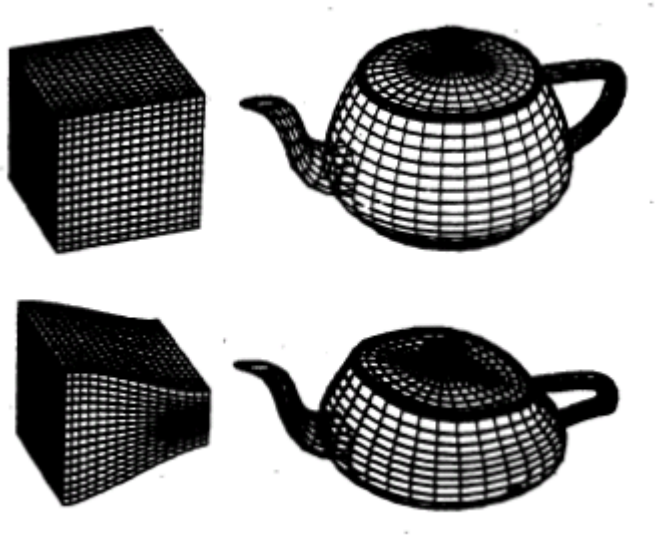
3) Faça uma pesquisa sobre as transformações de *tapering*, *twisting* e *bending*. Apresente figuras que exemplificam cada uma destas transformações (1.0 ponto).

Tapering, *twisting* e *bending* são transformações geométricas que no caso geral são não lineares.

A transformação de *tapering* consiste de uma escala em que o fator escala depende de uma das coordenadas. Por exemplo, um *tapering* 3D em coordenadas euclidianas de ordem quadrática é dado pela seguinte expressão, onde o fator de escala é a função $f(z) = 1+za_0+z^2a_1$:

$$Tp(\vec{X}) = \begin{bmatrix} f(z) & 0 & 0 \\ 0 & f(z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Isto causa um efeito de escala nas direções x e y que variam com a coordenada z, no caso por uma função quadrática. Na figura abaixo, observe como a escala é atenuada não linearmente conforme andamos em uma das direções

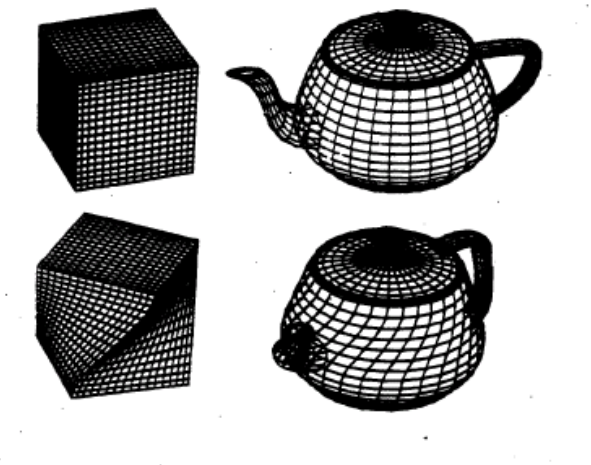


A transformação de *twisting* é um tipo de transformação em que os pontos do espaço são rotacionados em torno de um eixo, por um ângulo de rotação que varia em função da coordenada correspondente ao eixo. Por exemplo, a transformação abaixo

rotaciona os pontos do espaço de um ângulo teta = f(Z) em torno do eixo z. Observe como o ângulo de rotação é dado por uma função da coordenada z.

$$T(\vec{X}) = \begin{bmatrix} \cos(f(z)) & -sen(f(z)) & 0 \\ sen(f(z)) & \cos(f(z)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Na figura do cubo, o topo, cuja coordenada z é maior, sofre maior rotação que a base.



Finalmente, a transformação de *bending*, é uma transformação que tem o efeito de deformar um objeto ao longo de uma direção, dobrando-o. Sua equação é bem mais complexa que as duas anteriores. Consideremos um *bending* ao longo de uma direção paralela ao eixo Y. O *bending* depende de um ângulo de deformação θ que é constante para y<ymin e y>ymax, sendo respectivamente igual a $k(y\min - y_0)$ e $k(y\max - y_0)$.O valor k é a taxa da deformação e é medido em radianos e y_0 é o centro da deformação. Dentro do intervalo [ymin,ymax], $\theta = k(y - y_0)$ é uma função de y, o que faz com que a deformação ocorra não linearmente.

$$\theta = k(\hat{y} - y_0)$$
$$S_\theta = sen(\theta)$$
$$C_\theta = \cos(\theta)$$

$$\hat{y} = \begin{cases} y_{\min}, y < y_{\min} \\ y, y_{\min} < y < y_{\max} \\ y_{\max}, y > y_{\max} \end{cases}$$

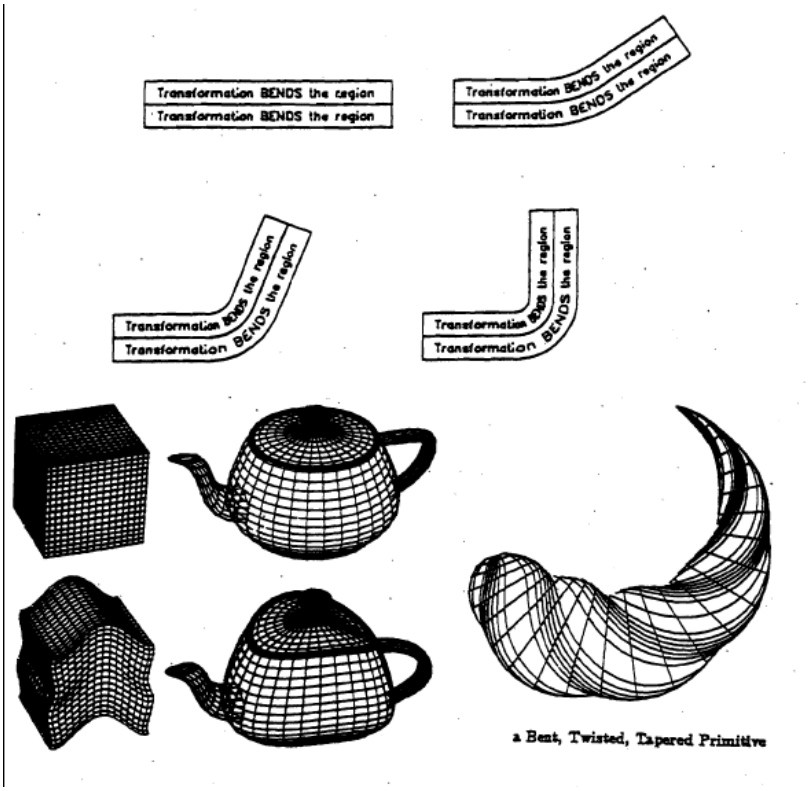
A deformação faz com que as novas coordenadas sejam dadas por

$$X = x$$

$$Y = \begin{cases} -S_\theta(z - \frac{1}{k}) + y_0, y_{\min} < y < y_{\max} \\ -S_\theta(z - \frac{1}{k}) + y_0 + C_\theta(y - y_{\min}), y < y_{\min} \\ -S_\theta(z - \frac{1}{k}) + y_0 + C_\theta(y - y_{\max}), y < y_{\max} \end{cases}$$

$$Z = \begin{cases} C_\theta(z - \frac{1}{k}) + \frac{1}{k}, y_{\min} < y < y_{\max} \\ C_\theta(z - \frac{1}{k}) + \frac{1}{k} + S_\theta(y - y_{\min}), y < y_{\min} \\ C_\theta(z - \frac{1}{k}) + \frac{1}{k} + S_\theta(y - y_{\max}), y < y_{\max} \end{cases}$$

Como pode ser visto, a coordenada X permenece fixa e nos extremos, isto é, para y<ymin e y>max, os pontos são defomados por uma rotação mais uma translação. Na região interna ocorre uma deformação não linear. Observe, na figura do cubo, abaixo como o centro da figura é puxado na direção vertical enquanto as extremidades são curvadas para baixo.



Referências:
Figuras:
http://www.sccg.sk/~durikovic/classes/MRT/3DModeling2%20Transformations.pdf

Teoria:
Alan H. Barr. 1984. Global and local deformations of solid primitives. *SIGGRAPH Comput. Graph.* 18, 3 (January 1984), 21-30. DOI=10.1145/964965.808573 http://doi.acm.org/10.1145/964965.808573

4) Monte uma matriz de transformação que rotacione localmente um objeto deformando-o de tal modo que o ângulo de rotação é proporcional a coordenada z dos pontos do objeto. Que transformação é esta? (**Dica: ver a questão 3**) (1.0 ponto).

A transformação é um twisting.

$$T(\vec{X}) = \begin{bmatrix} \cos(z) & -sen(z) & 0 \\ sen(z) & \cos(z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

5) Explique em que tipo de consulta podemos utilizar o grafo dual, que descreve parte da estrutura topológica combinatória (noção de vizinhança discreta entre vértices, faces e arestas) de uma malha (1.0 ponto).

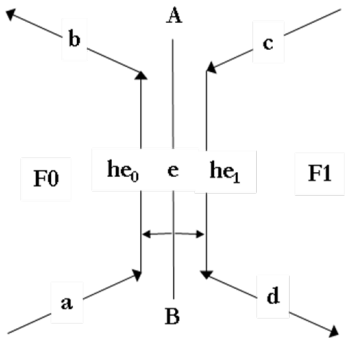
As consultas que requerem determinar a adjacência entre as faces ou polígonos da malha requerem o uso do grafo dual. Por exemplo, se quisermos determinar se duas faces são adjacentes não conseguiremos fazer isso simplesmente consultando as vizinhanças codificadas no grafo primal, isto é, aquele induzido pelas relações diretas entre os vértices e arestas da malha. O grafo primal normalmente é codificado por uma lista de vértices, arestas e faces. Para codificar um grafo dual de forma simples, podemos adicionar na lista de arestas, as duas faces que a ela são incidentes.

6) Faça uma pesquisa sobre a estrutura Half-Edge. Em que ela se distingue da estrutura Winged-Edge? (1.0 ponto).

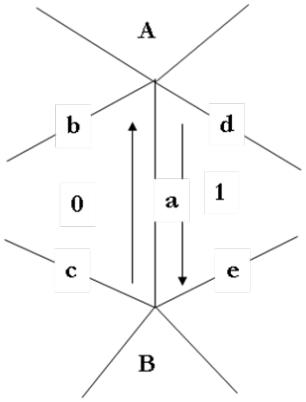
A estrutura Half-Edge é uma estrutura de dados topológica para representação de variedades bidimensionais que são generalizações de superfícies. Grosso modo, podemos dizer que elas são estruturas capazes de representar malhas nas quais cada aresta somente possui somente duas faces que sobre ela incidem (somente uma se for uma aresta do bordo da malha).

Na estrutura Half-Edge cada aresta ***e*** é tratada como um par de semi-arestas, as half-edges (na figura, h_0 e h_1). Cada half-edge armazena uma referência para sua face

incidente e referências para as half-edges sucessoras e predecessoras na face considerada. Cada half-edge também mantém uma referência para a sua half-edge gêmea.



Diferentemente, na estrutura Winged-Edge, cada aresta mantém uma referência para as faces incidentes à direita e à esquerda. Também mantém referências para as arestas sucessoras e predecessoras no sentido de percorrimento anti-horário nas faces esquerda e direita.



aresta	vértice1	vértice2	face esq	face dir	pred esq	succ esq	pred dir	succ dir
a	B	A	0	1	c	b	d	e

7) Considere uma malha representada por uma lista de faces, arestas e vértices. Descreva um algoritmo para enumerar todas as faces incidentes a um vértice v (1.0 ponto).

Algoritmo RetornaFacesIncidentes(v,LF,LV,LE): F
Entrada
v - vértice expresso por suas coordenadas
LF- lista de faces
LV- lista de vértices
LE- lista de arestas
Saída:
Uma lista de faces incidentes a v
Inicio

```
F ←{}
{Descobrimos o índice do elemento da lista de faces com as mesmas coordenadas de v}
Para i = 0 até Tamanho(LV)-1 faça
        Se (Coordenadas(LV[i]) = Coordenadas(v))
                indiceLV ← i
        Fim_Se
Fim_Para

{ Para cada face da lista de faces}
Para i=0 até Tamanho (LF)-1 faça
        {Para cada aresta da face corrente de índice i}
        Para j = 0 até Tamanho (LF[i])-1 faça
                {Pegue o índice da aresta j da face i na lista de arestas}
                indiceLE = LF[i][j];
                aresta = LE[indiceLE];
                {Se o índice do primeiro ou segundo vértice da aresta for igual ao índice
                do vértice desejado v}
                Se ( aresta[0] = indiceLV ou aresta[1] = indice LV) então
                        {Adicione a face na lista de faces incidentes a v}
                        F = F ∪ LF[i];
                Fim_Se;
        Fim_Para;
FimPara;

Retorne F;
```

Fim_Algoritmo

8) Considere uma malha representada por uma lista de faces, arestas e vértices. Descreva um algoritmo para enumerar todas as arestas incidentes a um vértice (1.0 ponto).

Algoritmo RetornaArestasIncidentes(v,LF,LV,LE): A
Entrada
v - vértice expresso por suas coordenadas
LF- lista de faces
LV- lista de vértices
LE- lista de arestas
Saída:
Uma lista de arestas incidentes a v
Inicio
{Descobrimos o índice do elemento da lista de faces com as mesmas coordenadas de v}
A ← {}
Para i = 0 até Tamanho(LV)-1 faça
    Se (Coordenadas(LV[i]) = Coordeadas(v))
        indiceLV ← i
    Fim_Se
Fim_Para

{ Para cada aresta da lista de faces}
Para i=0 até Tamanho (LE)-1 faça
    aresta = LE[i];
    {Se o índice do primeiro ou segundo vértice da aresta for igual ao índice
    do vértice desejado v}
    Se (aresta[0] = indiceLV ou aresta[1] = indice LV) então
        {Adicione a aresta na lista de arestas incidentes a v}
        A = A ∪ aresta;
    Fim_Se;
Fim_Para;

Retorne A;

Fim_Algoritmo

9) Considere a expressão para uma curva de Bézier cúbica C(u). Apresente a expressão análoga para um retalho de Bézier cúbico S(u,v) (1.0 ponto).

A expressão é simples. Considerando P(i,j) a matriz de pontos de controle, basta tomar o produto tensorial das bases de Bézier cúbicas nas direções de u e v formando a expressão abaixo

$$S(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(u)B_j^3(v)P(i,j)$$

10) Explique o que é uma curva Nurbs (1.0 ponto).

Uma Nurbs é uma variação da curva B-Spline não-uniforme, isto é, onde a distância entre os valores de nó não é a mesma, dada pela razão de dois polinômios.

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

Os valores w_i associados a cada ponto de controle são pesos que podem ser vistos como parâmetros extras. Os pesos w_i afetam a curva apenas localmente, sendo a curva atraída para um ponto P_i se o w_i correspondente aumenta e é afastada de P_i se w_i diminui. Pode-se compreender os pesos w_i como parâmetros de acomplamento da curva aos pontos de controle.