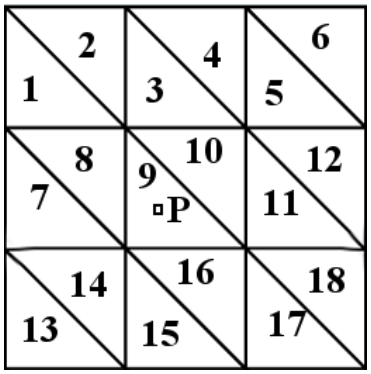
P2 CG 2011.2

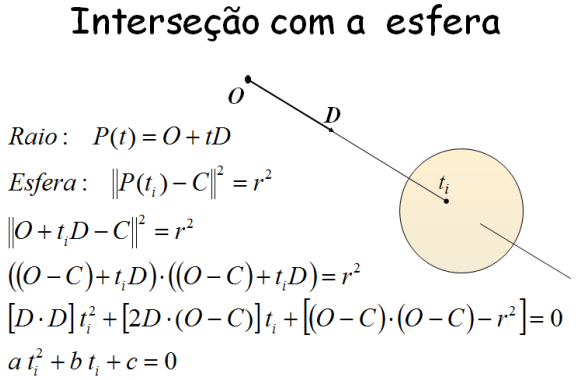
	Computação Gráfica 1 Prof. Rodrigo de Toledo	Data: 16/11/2011 P2

1) (1,5 pontos) Qual a ordem correta no pipeline gráfico entre as seguintes etapas?
(a) Clipping; (b) Display/Visibility; (c) Illumination (shading); (d) Modeling Transformations; (e) Projection; (f) Scan Conversion (rasterization); (g) Viewing Transformation.

2) (1,0 pontos) Em qual dessas etapas acima o algoritmo de Bresenham é exaustivamente usado e por que?

3) (2,5 pontos) Dada a malha abaixo, e as normais de cada triângulo (n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18). Considere três diferentes tipos de shading (a) Flat, (b) Gouraud, (c) Phong; para cada um (a,b,c) diga como (e quais) as normais serão usadas para calcular a cor final do pixel P.





4) (2,5 pontos) No algoritmo de raytrace, às vezes não se deseja saber o ponto de interseção, mas apenas se houve interseção (útil para teste de sombra por exemplo, ou para bound-sphere de um objeto mais complexo). Qual é o teste (verdadeiro/falso) mais simples que devemos fazer para retornar apenas se houve interseção ou não entre um raio e uma esfera, usando a equação acima? (inspirada na questão 15.19 do Foley)

5) (2,5 pontos) O que faz o vertex shader abaixo, se aplicado à malha de uma esfera, centrada em zero e raio um (como a "ST Sphere" do ShaderLabs)? Qual o resultado visual? O que deve ser alterado para que a normal funcione em todos os casos?

```
void main() {
vec4 v = gl_Vertex;
vec3 normal = vec3(0.0,0.0,0.0);
vec3 color;
float lado = 0.5;
if(v.x > lado){
        v.x = lado;
       color = vec3(0.0,0.0,1.0);
       normal += vec3(1.0,0.0,0.0);
} else if(v.x < -lado){
       v.x = -lado;
       color = vec3(0.0,1.0,0.0);
}
if(v.y > lado){
        v.y = lado;
       color = vec3(1.0,0.0,1.0);
} else if(v.y < -lado){
       v.y = -lado;
       color = vec3(1.0,1.0,1.0);
}
```

```
if(v.z > lado){
        v.z = lado;
       color = vec3(1.0,0.0,0.0);
} else if(v.z < -lado){
       v.z = -lado;
       color = vec3(1.0,1.0,0.0);
}

normal = normalize(normal);
vec3 dir_luz = gl_LightSource[0].position.xyz;
dir_luz = normalize(dir_luz);
float difusa = max(dot(normal, dir_luz),0.0);;
gl_FrontColor.xyz = color*difusa + color*0.2;
gl_FrontColor.w = 1.0;
gl_Position = gl_ModelViewProjectionMatrix * v;
}
```