MAC420/5744: Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #2: Conceitos

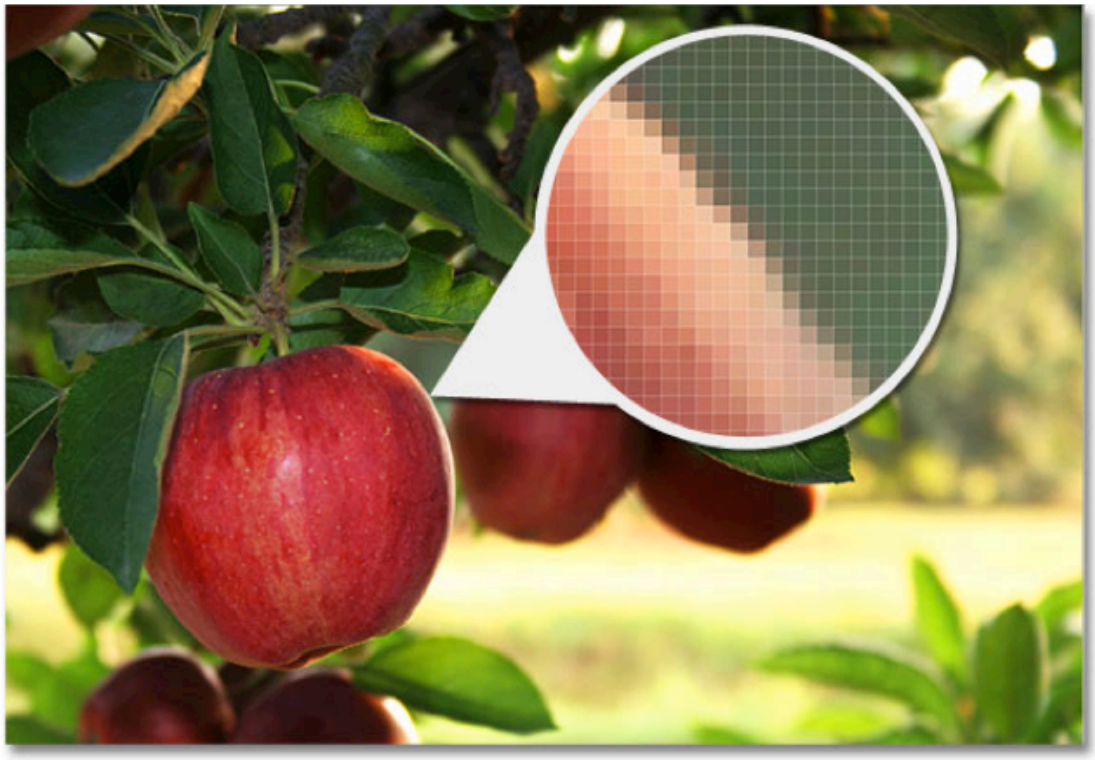
Some slides by Steve Marschner, Cornell CS4620/5620

Computação gráfica

- A computação gráfica trata de todos os aspectos relacionados à criação ou síntese de imagens com um computador:
  - Hardware
  - Software
  - Aplicações

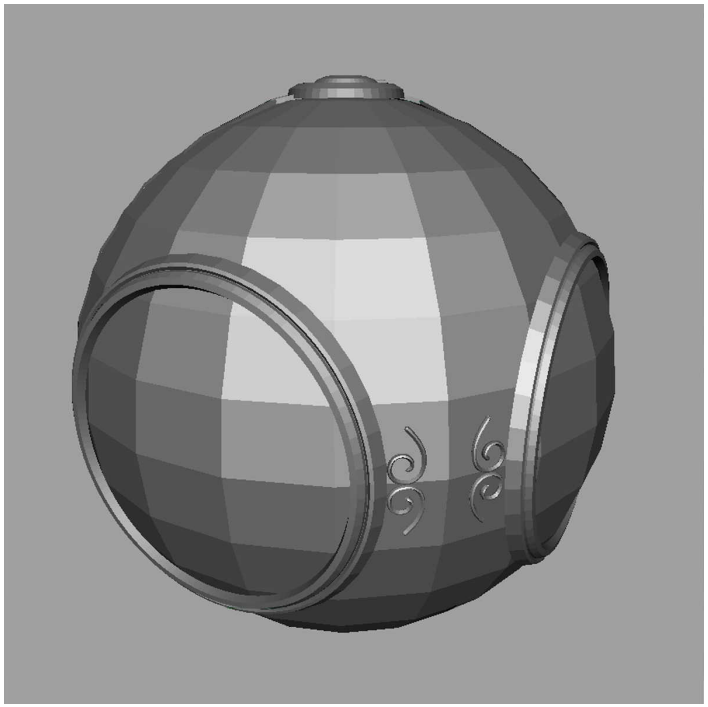
Gráficos "raster"

- Imagem produzida por um array (ou "raster") de pixels ("picture elements") no frame buffer



Gráficos “raster”

- Deixamos o mundo das “retas” ou modelos em wireframe para chegar em polígonos preenchidos



Computação gráfica interativa

- Controle do conteúdo, estrutura, e a aparência de objetos e suas imagens através de feedback visual
- Componentes de um sistema interativo de CG:
  - Entrada (e.g. mouse, tablet e stylus, multi-touch, etc)
  - Processamento (e armazenamento)
  - Display/Saída (e.g. tela, impressora, gravador de vídeo, etc)

O primeiro sistema deste tipo foi o "Sketchpad", desenvolvido por Ivan Sutherland em 1963, durante o seu doutorado no MIT.

Note o monitor CRT, a caneta ótica e o painel multifunção.

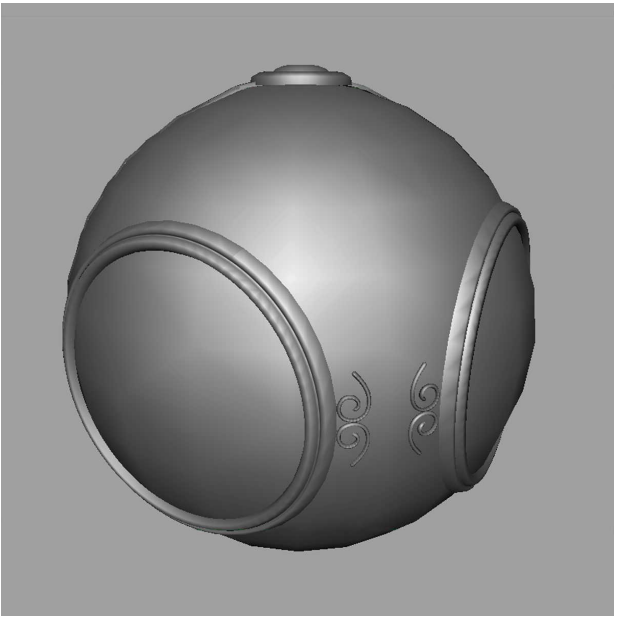
http://youtu.be/J6UAYZxFwLc

1970-1980

- Início dos padrões gráficos
  - FIPS ("Federal Information Processing Standard")
    - GKS ("Graphics Kernel System")
      - Tornou-se padrão ISO 2D - Europa
  - Core: iniciativa norte-americana
    - 3D, mas não foi eleita como padrão ISO
- Gráficos "raster"
- Workstations e PCs

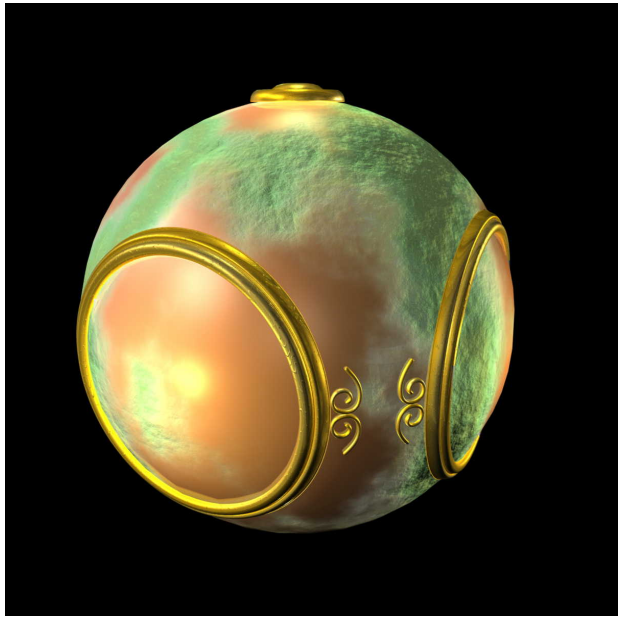
1980-1990

- Aumento do realismo



smooth shading

environment mapping



bump mapping

1980-1990

- Hardware especializado
  - Silicon Graphics "Geometry Engine"
    - Implementação do pipeline gráfico em VLSI
- Padrões da indústria
  - PHIGS (3D)
  - RenderMan (Pixar)
- Gráficos em rede: X Window System
- Interface Homem-Computador (HCI)

SGI Extreme 4GE7MCM (SGI Indigo)

1990-2000

- OpenGL API
- Filmes inteiramente gerados por computador (e.g. Toy Story, 1995)
- Novas capacidades em hardware
  - Mapeamento de textura
  - Blending
  - Buffers de acumulação e estêncil

2000-2010

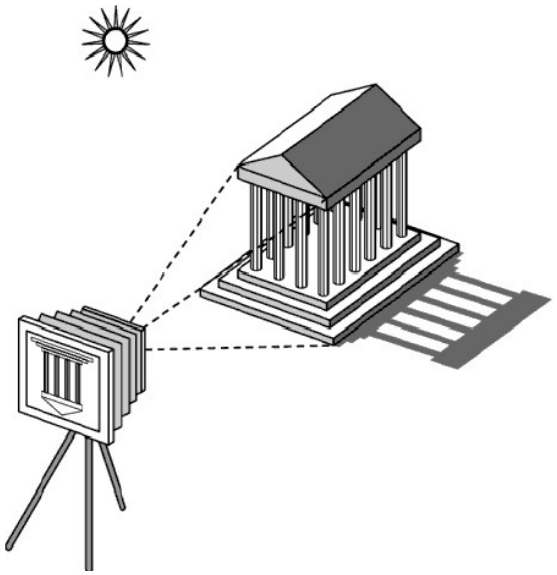
- Fotorealismo
- Placas gráficas para PC dominam o mercado:
  - NVidia, AMD (ex ATI), Intel
  - Pipelines programáveis
    - GPU
- Game boxes (videogames) determinam a direção do mercado
- CG se torna rotina em produções cinematográficas: Maya, Lightwave
- Novas tecnologias de display

2011-atual

- CG é ubíqua
  - Telefones celulares
  - Dispositivos embarcados
- OpenGL ES e WebGL
- Realidade alternativa e aumentada
- TVs e filmes em 3D

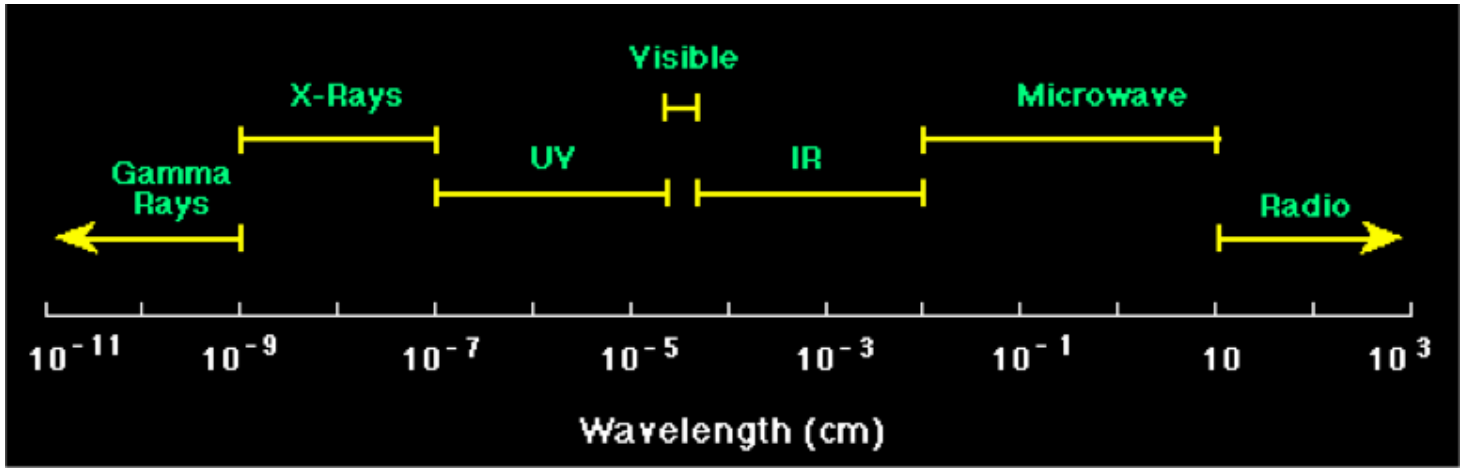
Formação da imagem

- Usa um processo análogo aos formadas por sistemas de imagem físicos (e.g. câmeras, microscópios, sistema visual humano)
- Elementos da formação da imagem
  - Objetos
  - Observador (Viewer)
  - Fonte(s) de iluminação



Luz

- A luz é parte do espectro eletromagnético que é sensível ao sistema visual humano
- Ondas eletromagnéticas que têm tamanho variando entre 350 e 750 nanômetros
- Ondas maiores aparecem como vermelho e menores como azul (infra-vermelho e ultra-violeta)

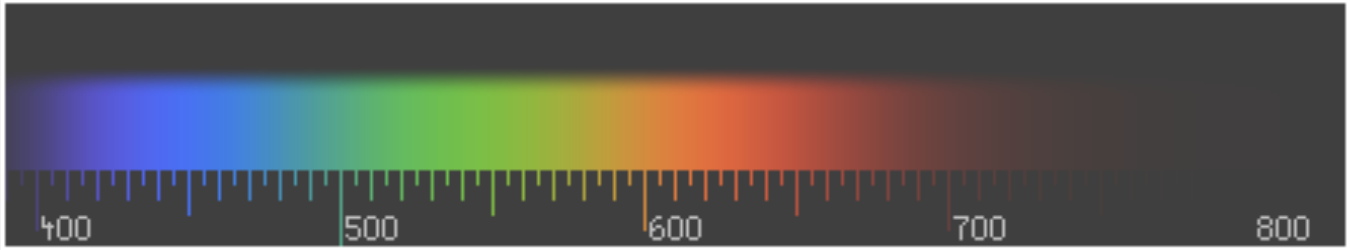


Espectro visível

Cores do espectro visível

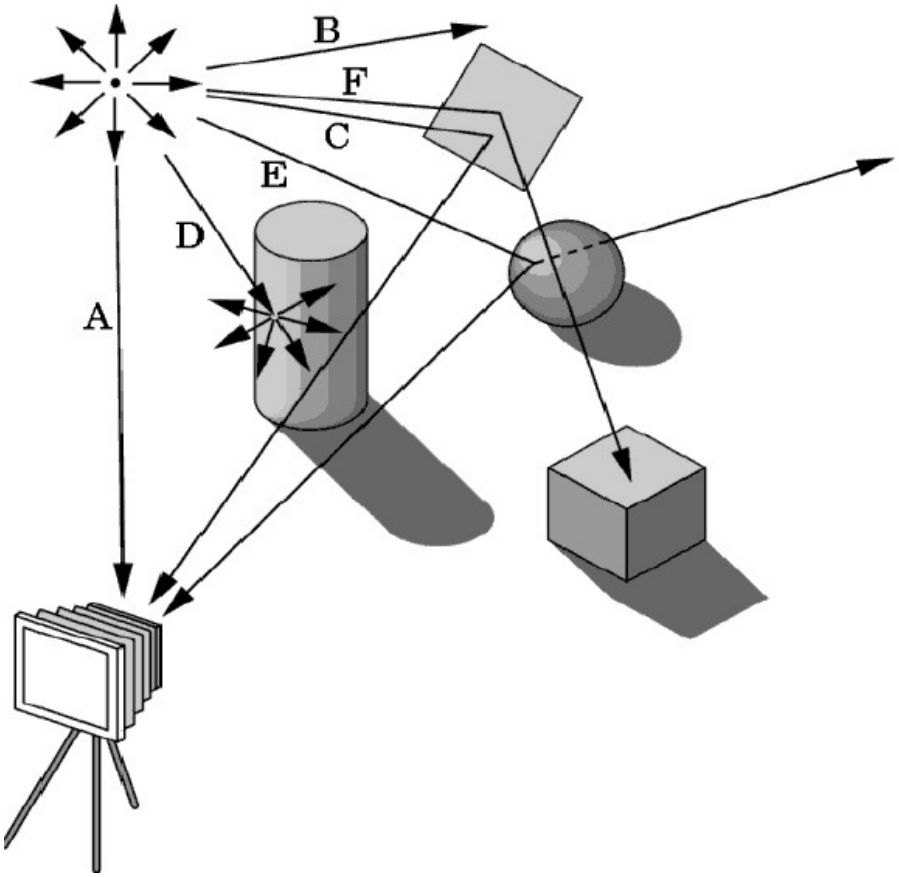
Cor	Comprimento de onda	Freqüência
vermelho	~ 625-740 nm	~ 480-405 THz
laranja	~ 590-625 nm	~ 510-480 THz
amarelo	~ 565-590 nm	~ 530-510 THz
verde	~ 500-565 nm	~ 600-530 THz
ciano	~ 485-500 nm	~ 620-600 THz
azul	~ 440-485 nm	~ 680-620 THz
violeta	~ 380-440 nm	~ 790-680 THz

Espectro Contínuo



Ray tracing

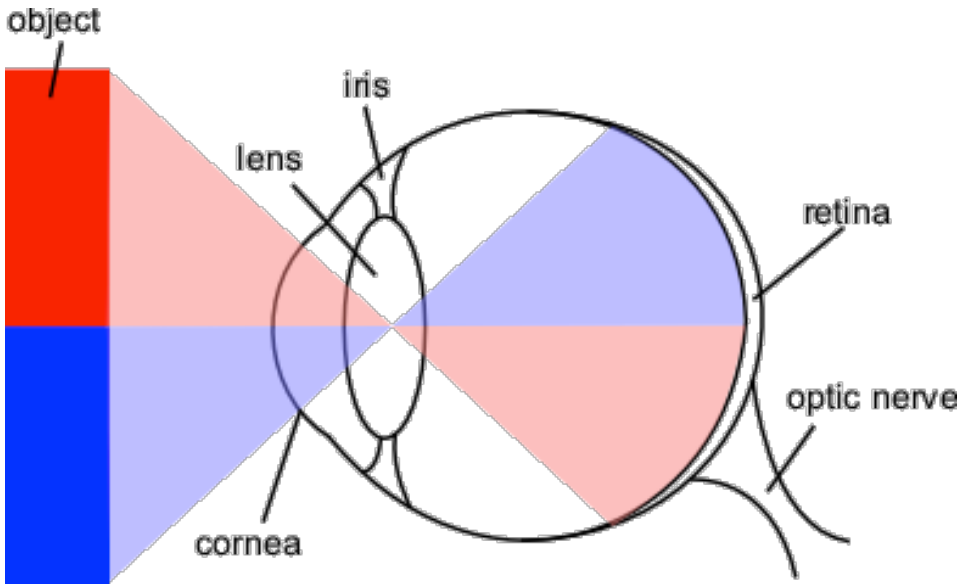
- Uma maneira de formar uma imagem é seguir raios de luz encontrando quais desses raios entram nas lentes da câmera
- Entretando, cada raio de luz pode ter múltiplas interações com objetos antes de ser absorvido ou ir para o infinito.



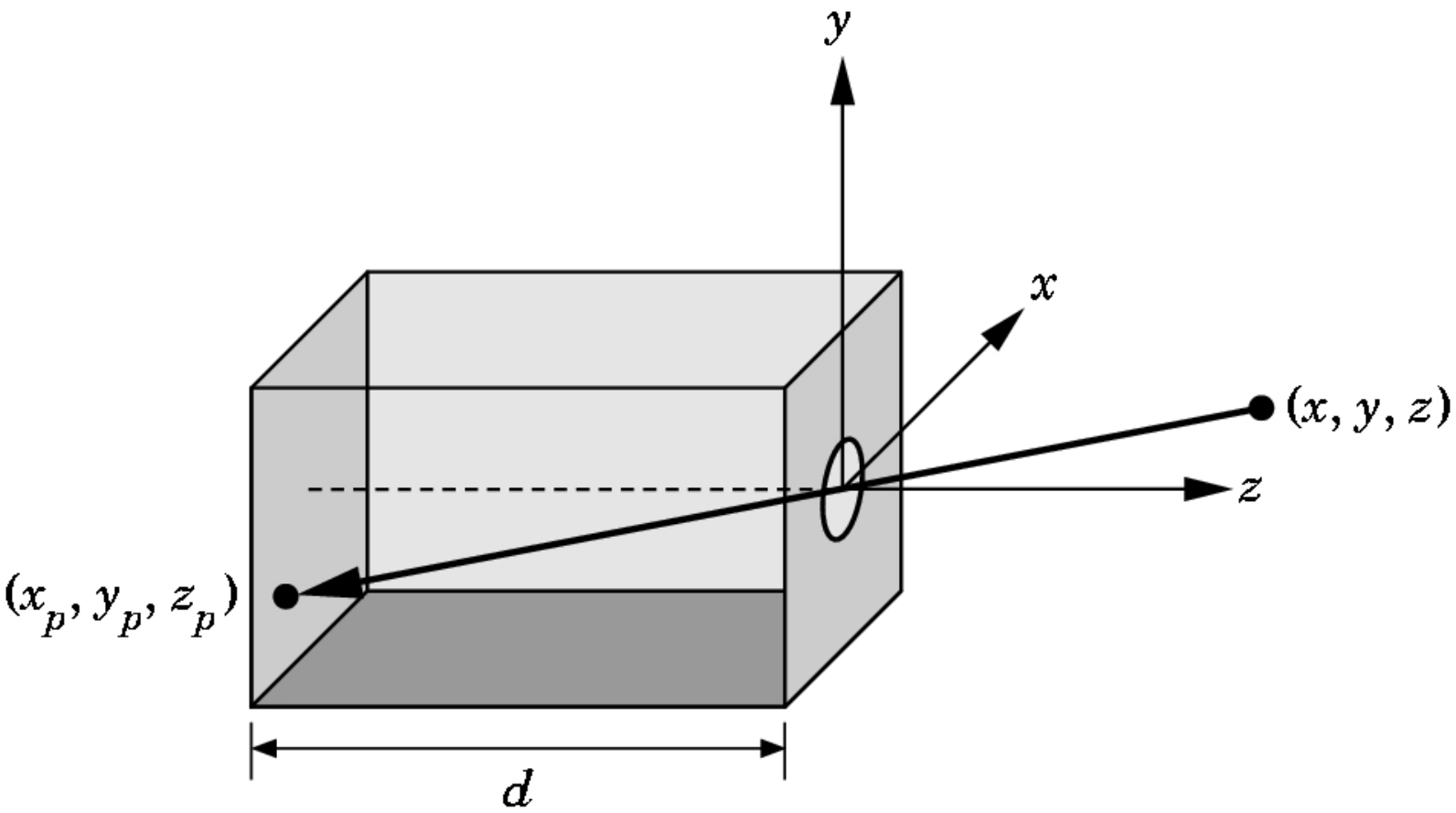
Ray tracing

Sistema visual humano

- Tem dois tipos de sensores
- Bastonetes
  - Monocromáticos, visão noturna
- Cones
  - Sensível a cores
  - Três tipos de cones
- Processamento inicial da luz em humanos segue os mesmos princípios da maioria dos sistemas óticos.

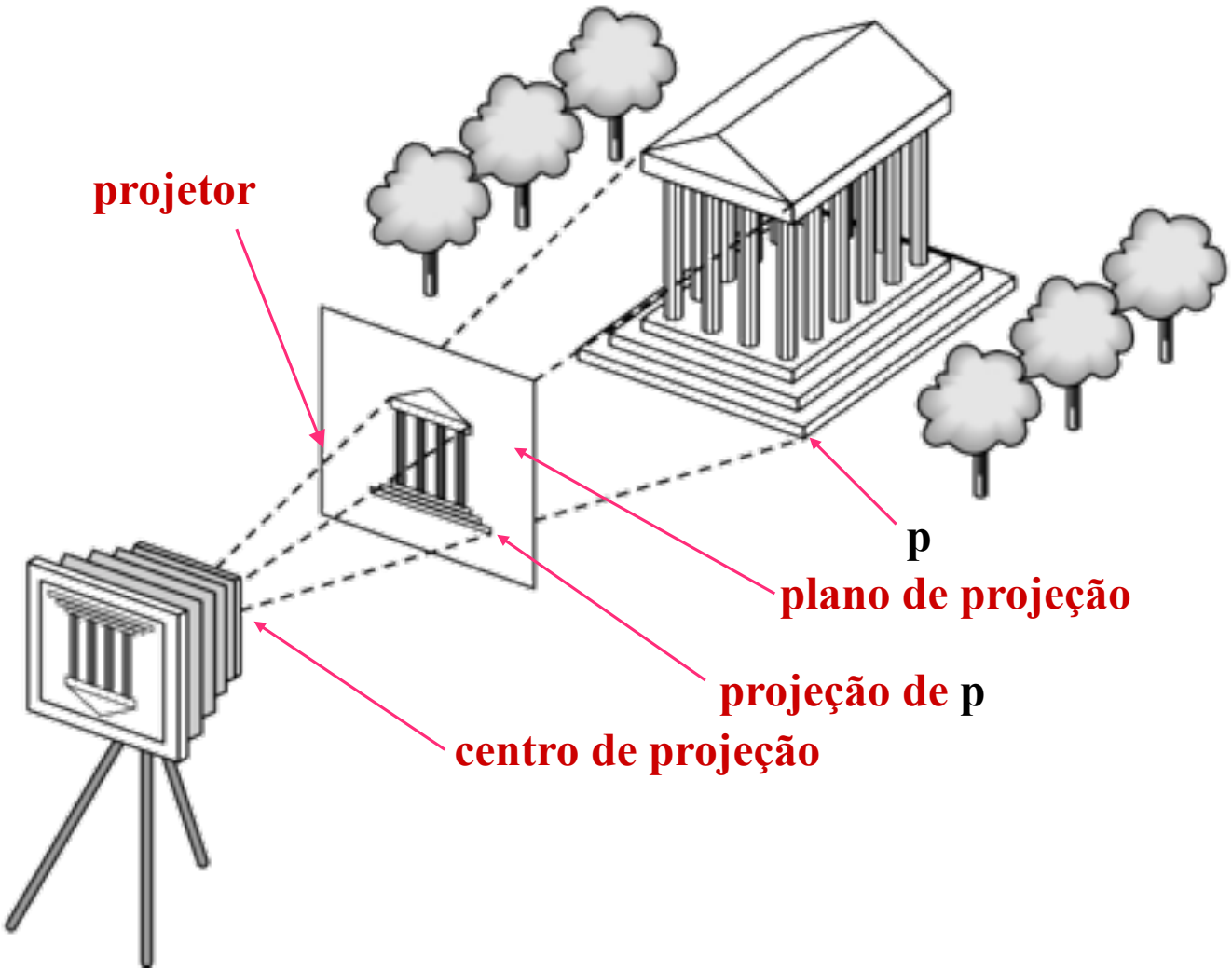


Câmera "pinhole"



$x_p = -x/z/d$ $y_p = -y/z/d$ $z_p = d$

Modelo de câmera

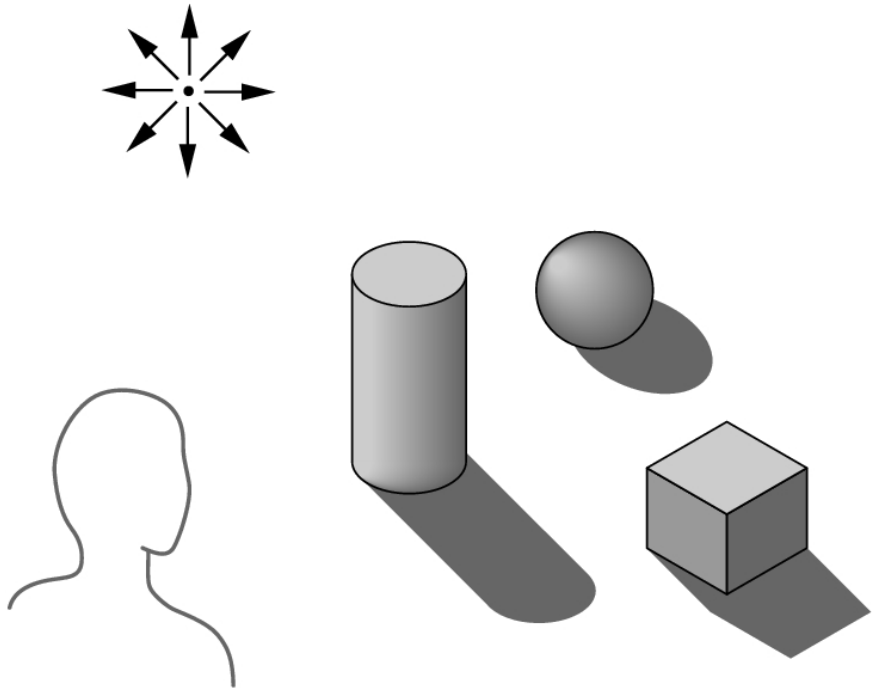


Vantagens

- Separação entre objetos, observador e fontes de iluminação
- Gráficos 2-D tornam-se um caso especial dos gráficos 3-D
- Implica em uma API software mais simples
  - Especificação de objetos, luzes, câmera e atributos
  - A implementação determina a imagem
- Implementação rápida em hardware

Iluminação global vs. local

- Não podemos simplesmente calcular a cor e o sombreamento de um objeto de forma independente
  - Alguns objetos são bloqueados da luz
  - A luz pode refletir de objeto a objeto
  - Alguns objetos pode ser translúcidos



Ray tracing - desvantagens

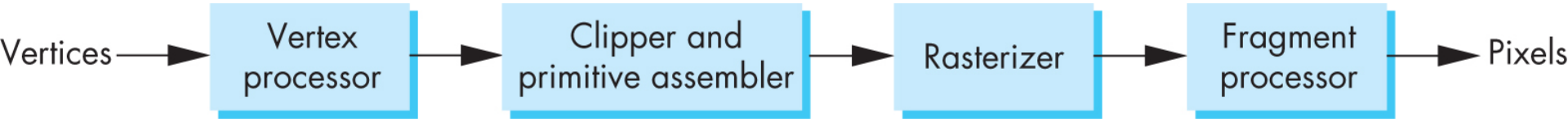
- Ray tracing parece modelar a formação de imagens utilizando conceitos físicos reais, por que então não usá-lo em um sistema gráfico?
  - É possível de ser implementado para objetos geométricos simples constituídos de polígonos e superfícies quádricas
  - Em princípio, pode-se reproduzir efeitos como sombras e reflexões múltiplas porém ainda é um processo lento, o que dificulta o seu uso em aplicações interativas.

Definição de uma API gráfica

- Como podemos utilizar o modelo de câmera sintética para criar aplicações em computação gráfica ?
  - Application Programmer Interface (API)
- Precisamos somente especificar
  - Objetos
  - Materiais
  - Observador
  - Luzes
- Como seria a sua implementação ?

Abordagem prática

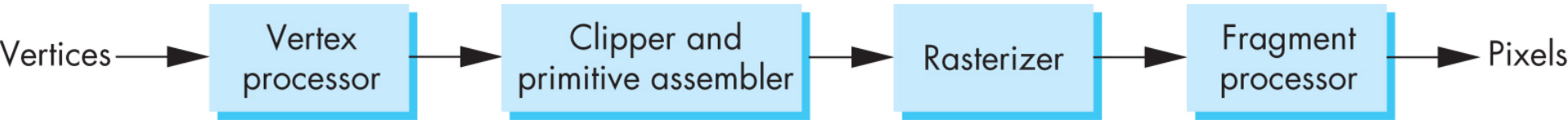
- Processa um objeto de cada vez, na ordem que são gerados pela aplicação
- Considera somente iluminação local
- Arquitetura do pipeline



- Todos estes passos podem ser implementados em hardware (ex: placa gráfica, GPU)

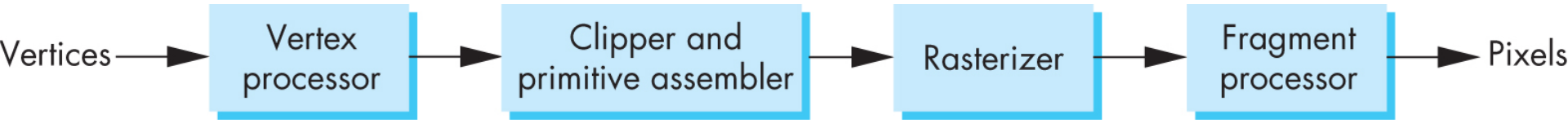
Processamento de vértices

- A maioria do trabalho executado no pipeline diz respeito à conversão entre sistemas de coordenadas
  - Coordenadas dos objetos
  - Coordenadas da câmera (observador)
  - Coordenadas da tela
- Cada mudança é equivalente à uma matriz de transformação
- Também calcula cores dos vértices



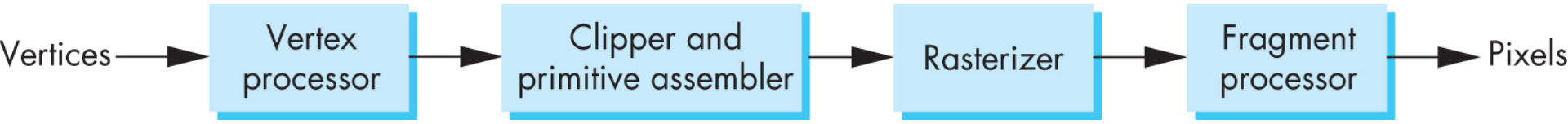
Projeções

- É o processo que combina o observador em 3D e objetos a fim de produzir uma imagem em 2D
  - **Projeção perspectiva:** todas as linhas de projeção se encontram no centro de projeção
  - **Projeção paralela**: linhas de projeção são paralelas, e o centro de projeção é substituído pela direção de projeção



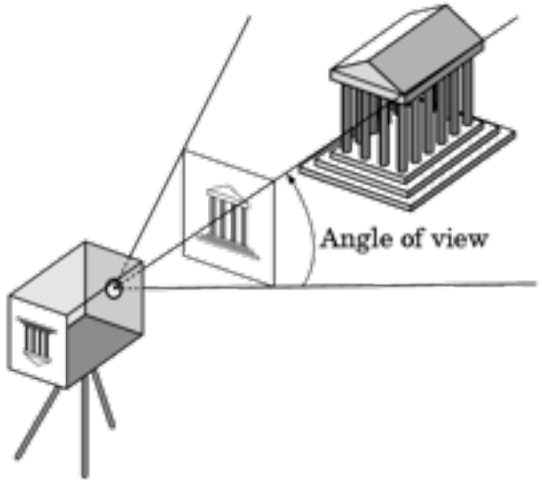
Agrupamento de vértices

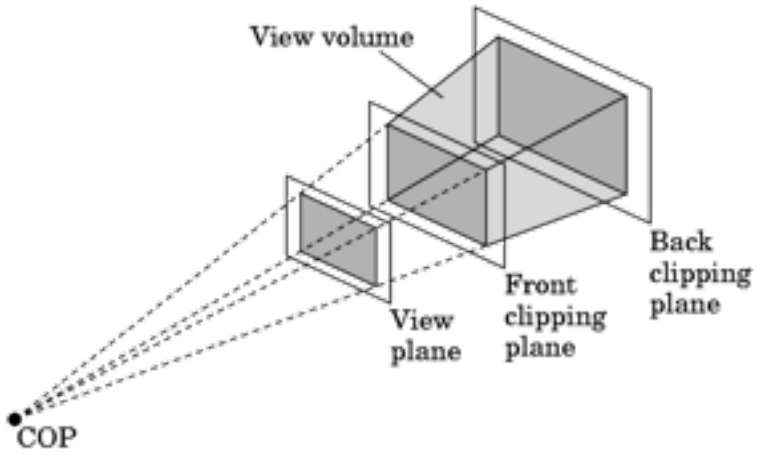
- Vértices devem ser agrupados em objetos geométricos antes das operações de recorte e rasterização
  - Segmentos de reta
  - Polígonos
  - Curvas e superfícies



Recortes

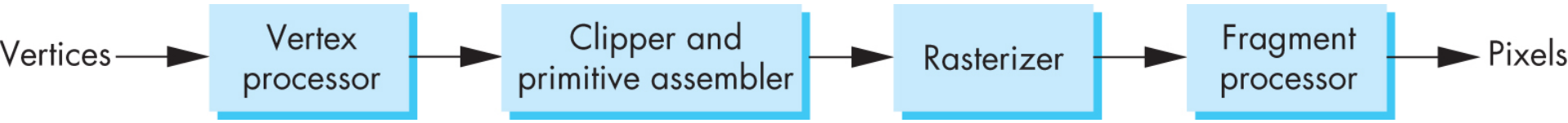
- Assim como uma câmera real não consegue "ver" todo o universo, uma câmera virtual pode somente enxergar parte dele também
- Objetos que não estejam neste volume são "recortados" da cena.





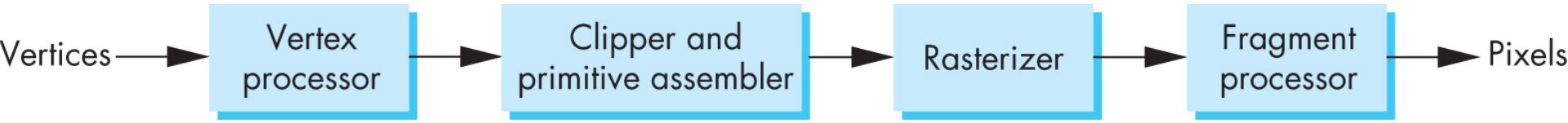
Rasterização

- Se um objeto não é recortado, cores devem ser atribuídas aos seus pixels
- O processo de rasterização produz uma série de fragmentos para cada objeto
- Fragmentos são ditos "pixels em potencial"
  - Possuem uma posição no frame buffer
  - Possuem atributos cor e profundidade
- O rasterizador interpola atributos dos vértices para toda a região do objeto.



Fragmentos

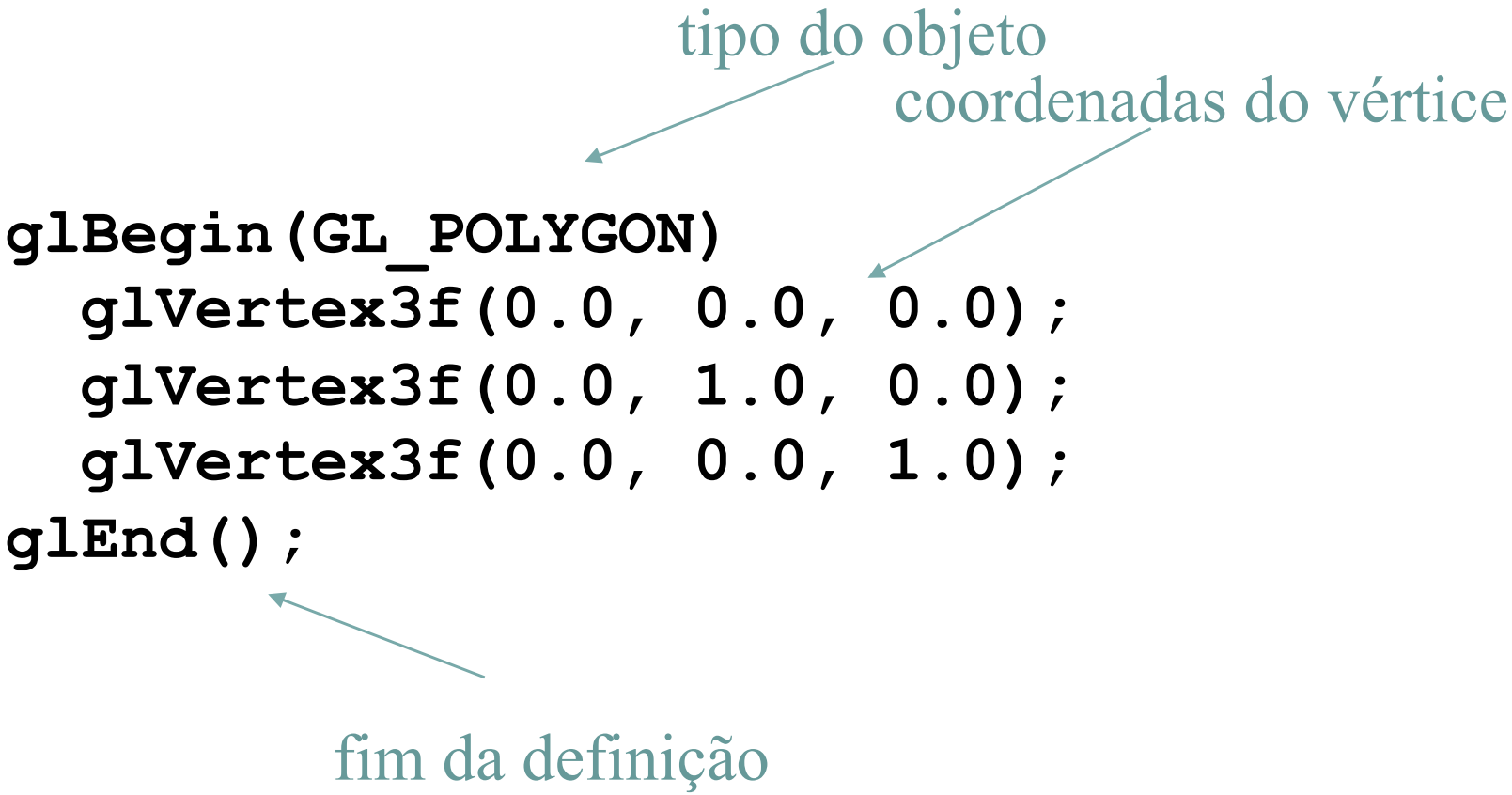
- Fragmentos são produzidos a fim de determinar a cor do pixel correspondente no frame buffer
- Suas cores são determinadas através de mapeamento de textura ou interpolação de cores entre vértices
- Fragmentos podem também estar sendo "bloqueados" por outros fragmentos mais próximos da câmera
  - Remoção de superfícies escondidas



Especificação de objetos

- A maioria das APIs suportam um número limitado de primitivas
  - Pontos (objeto 0D)
  - Segmentos de reta (1D objetos 1D)
  - Polígonos (objetos 2D)
- Algumas curvas e superfícies
  - Superfícies quádricas
  - Polinômios (paramétricos)
- São todos definidos através de coordenadas no espaço (vértices)

Exemplo OpenGL (antigo)



Exemplo OpenGL (atual)

- Armazena vértices em um array:

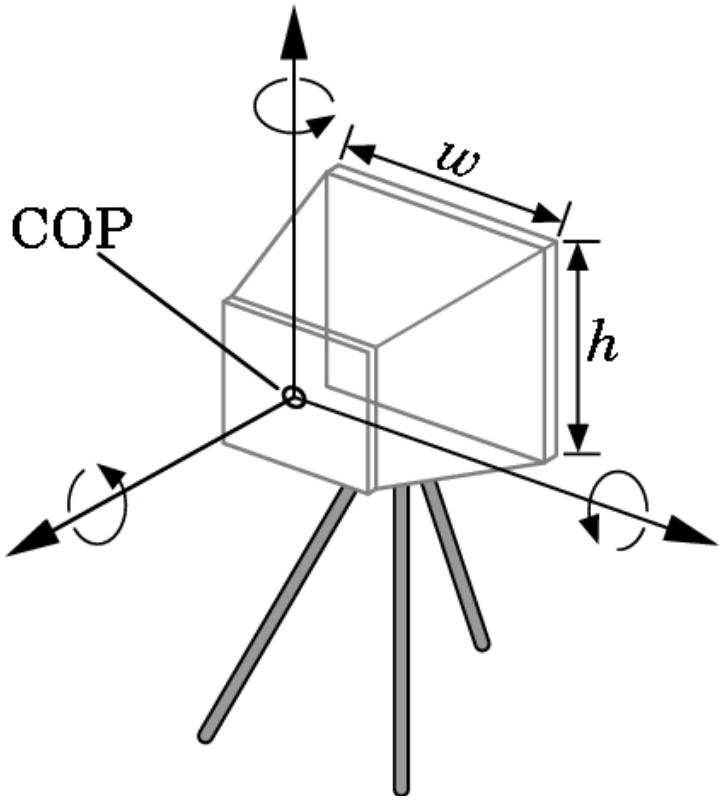
```
var points = [
 vec3(0.0, 0.0, 0.0),
 vec3(0.0, 1.0, 0.0),
 vec3(0.0, 0.0, 1.0),
];
```

```
var pts = new Float32Array(9);
pts[0]=0.0; pts[1]=0.0; pts[2]=0.0;
pts[3]=0.0; pts[4]=1.0; pts[5]=0.0;
pts[6]=0.0; pts[7]=0.0; pts[8]=1.0;
```

- Envia array para GPU
- Instrui GPU para renderizar como um triângulo

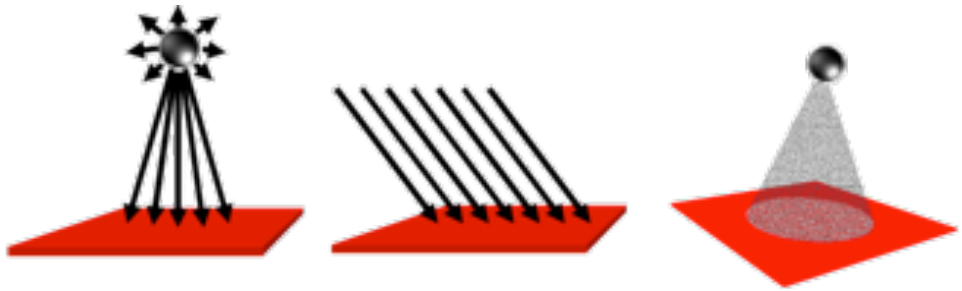
Especificação da câmera

- Seis graus de liberdade
  - Posição do centro da lente (COP)
  - Orientação
  - Lente
- Tamanho do filme
- Orientação do filme

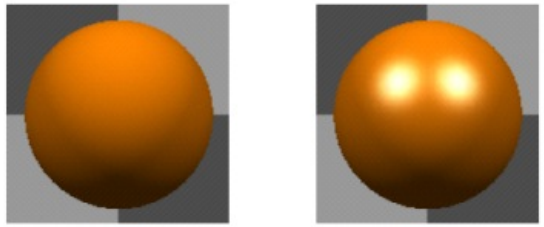


Luzes e materiais

- Tipos de luzes
  - Puntual
  - Direcional
  - Tipo "spot"



- Propriedades de materiais
  - Absorção: propriedades de cor
  - Reflexão
  - Difusa
  - Especular ("ponto de brilho")



OpenGL moderno

- Eficiência é alcançada utilizando GPU e não mais CPU
- Controle da GPU através de “shaders"
- A tarefa da aplicação é enviar dados e programas para a GPU
- GPU realiza a renderização

Modo “Immediate”

- A geometria é especificada por vértices
  - Posições em 2D ou 3D
  - Pontos, retas, círculos, polígonos, curvas, superfícies, etc.
- Modo “Immediate"
  - Toda vez que um vértice é especificado na aplicação, ele é enviado à GPU
  - O estilo antigo utiliza “glVertex"
  - Cria um gargalo entre a CPU e GPU
  - Removido a partir do OpenGL 3.1 e OpenGL ES 2.0

Modo "Retained"

- Coloca todos os vértices e seus atributos em arrays
- Envia-se array para a GPU renderizar imediatamente
- Solução razoável, porém precisamos enviar o array toda a vez que precisamos renderizar
- Então é melhor enviar o array e armazenar na memória da GPU para facilitar múltiplas renderizações.

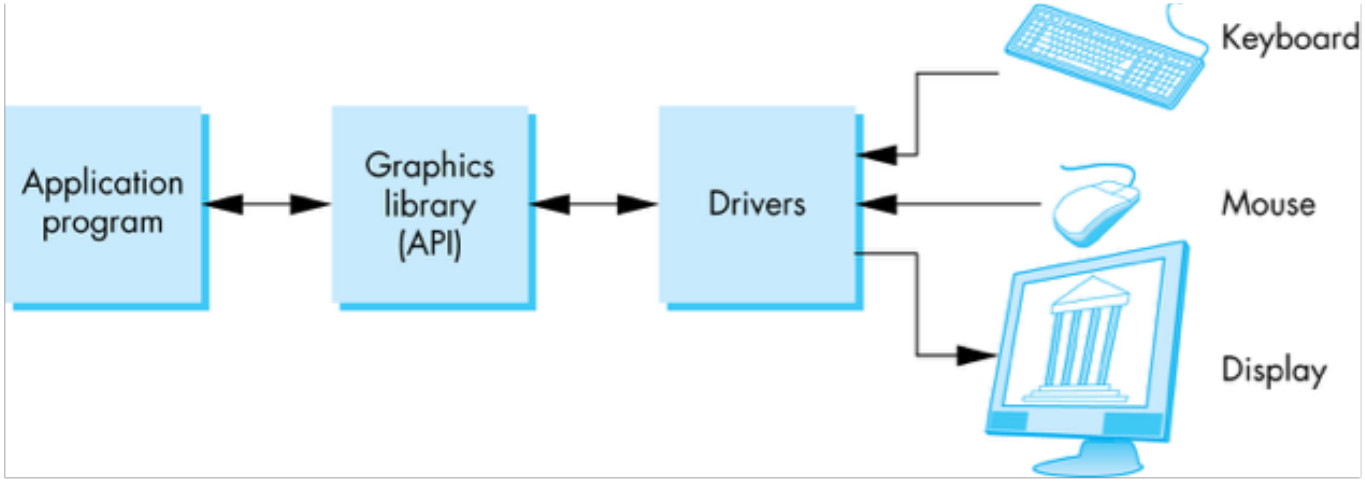
OpenGL >= 3.1

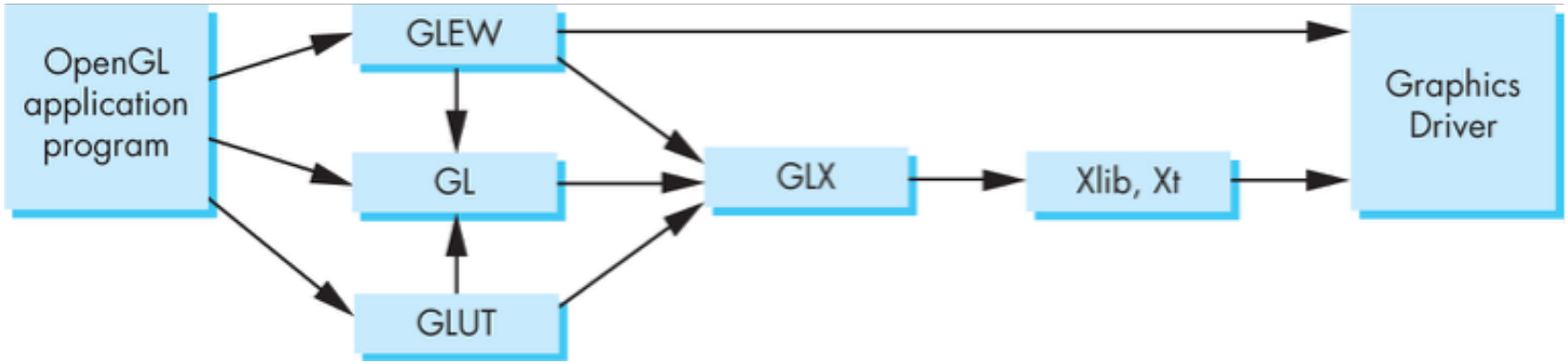
- Baseado totalmente em shaders
  - Não existe shader default
  - Cada aplicação deve prover um shader de vértice e outro de fragmento
- Não existe modo “Immediate"
- Poucas variáveis de estado
- Várias funções deprecadas

Outras versões

- OpenGL ES
  - Sistemas embarcados
  - Versão 1.0 - simplificação do OpenGL 2.1
  - Versão 2.0 - simplificação do OpenGL 3.1
    - Baseado em shaders
- WebGL
  - Implementação do ES 2.0 em JavaScript através do elemento Canvas do HTML5
  - Suportado nos browsers atuais
- OpenGL 4.1 e além
  - Adiciona outros shaders, como de geometria, tesselagem, cômputo

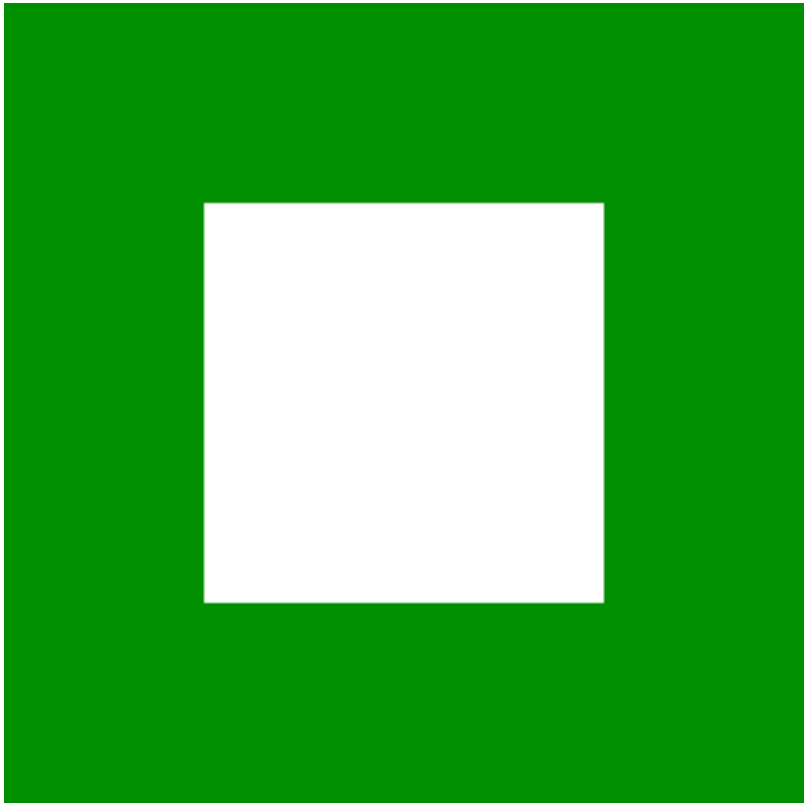
Organização do OpenGL



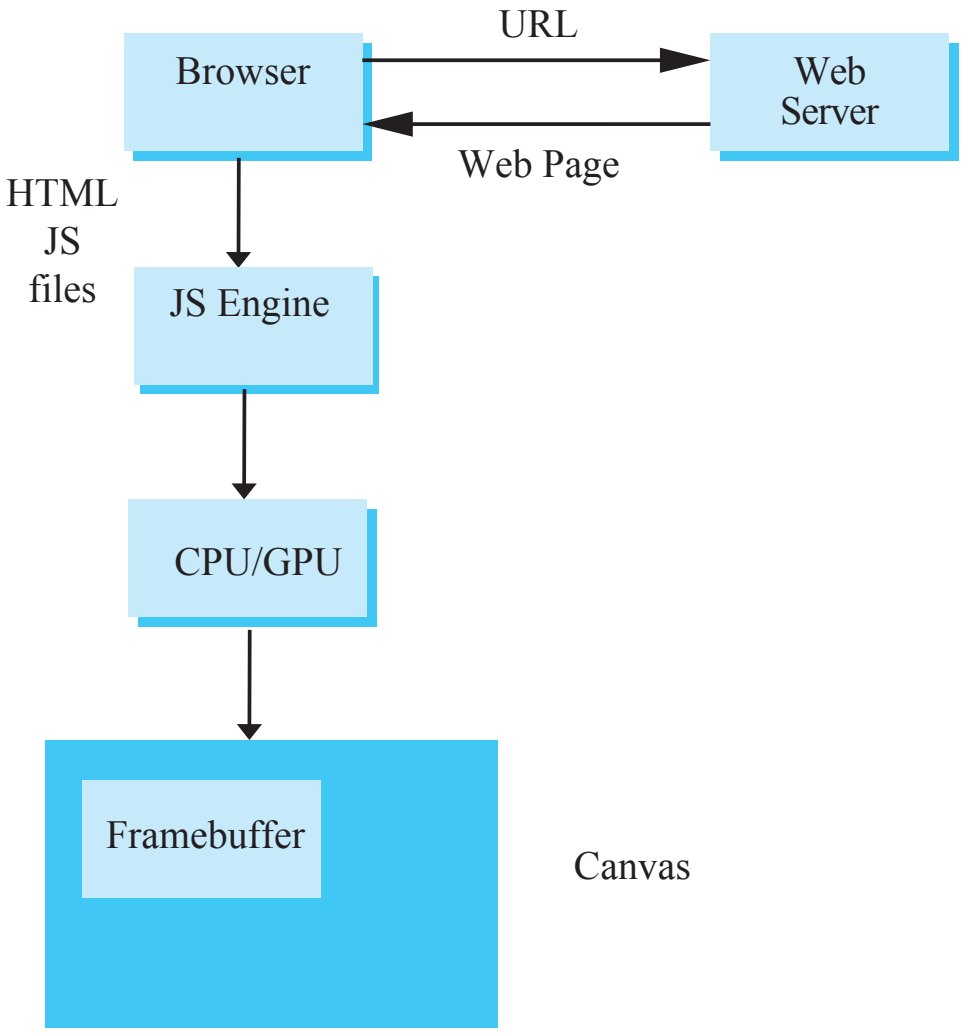


Exemplo

```
#include <GL/glut.h>

void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glClearColor(0.0,0.5,0.0,1.0);
    glutMainLoop();
}
```



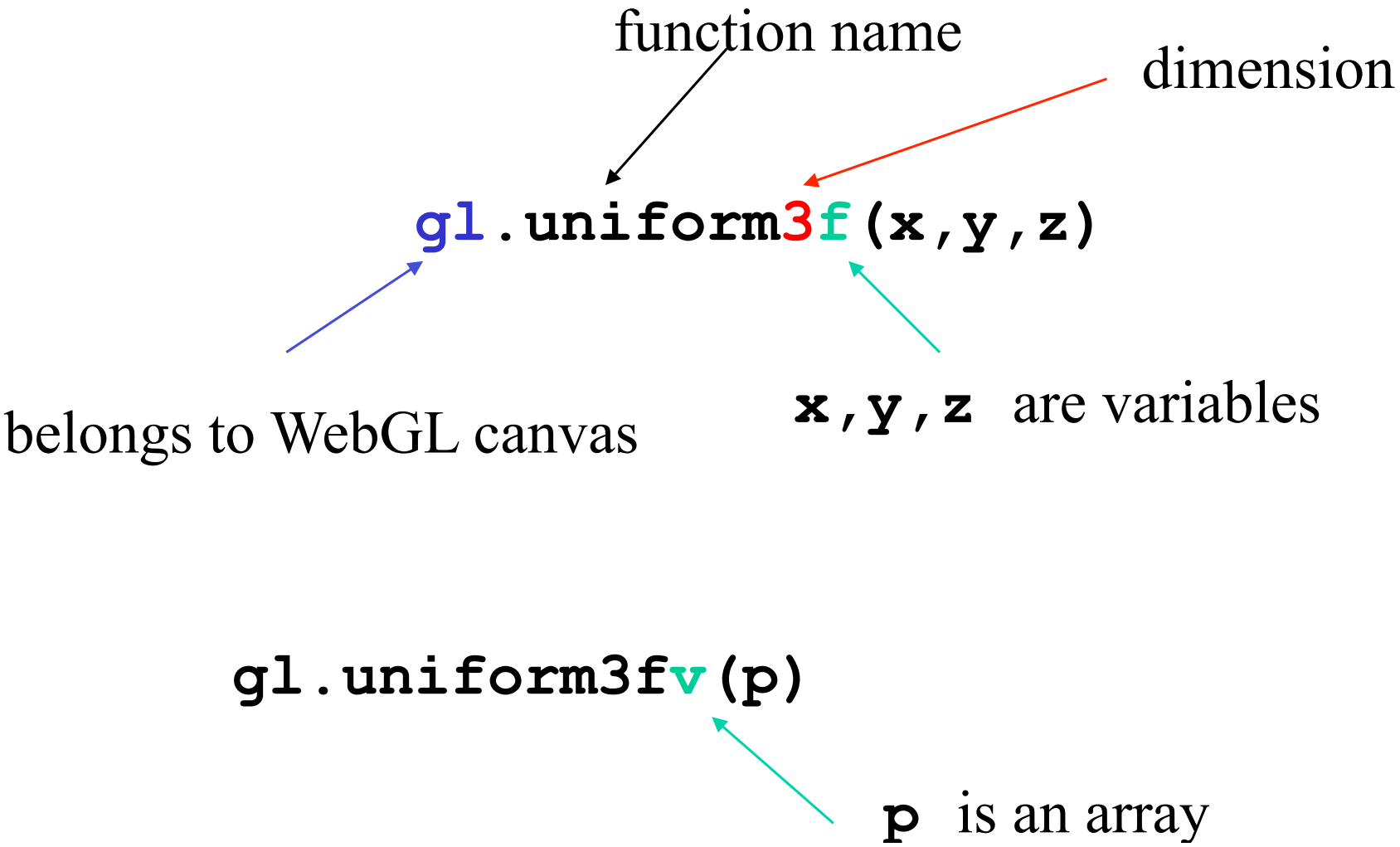
Execução no browser



Não é O-O

- Todas as versões do OpenGL nunca foram orientadas à objetos
- Múltiplas funções para cada tarefa específica
- Examplo: envio de valores à shaders
  - gluniform3f (gl.uniform3f)
  - gluniform2i (gl.uniform2i)
  - gluniform3dv (gl.uniform3dv)

Formato em WebGL



Constantes

- A maioria das constantes são definidas no objeto canvas
- No OpenGL desktop, elas estão nos arquivos #include (e.g. gl.h)
- Exemplos
  - OpenGL
    - glEnable(GL_DEPTH_TEST);
  - WebGL
    - gl.enable(gl.DEPTH_TEST);
    - gl.clear(gl.COLOR_BUFFER_BIT);

GLSL

- OpenGL Shading Language
- Parecida com a linguagem C
  - Vetores e matrizes (dimensões 2, 3 e 4)
  - Overloading
  - Construtores estilo C++
- Similar à Cg da Nvidia e HLSL da Microsoft
- Todo o código são enviados aos shaders como código fonte
- Eles então são compilados, ligados e então instanciados para execução

WebGL (5 passos)

1. Definição da página web (HTML file)
   - Requisitar um WebGL Canvas
   - Carregar quaisquer arquivos necessários
2. Definição de shaders (HTML file)
   - Pode ser feito em arquivo separado (depende do browser)
3. Especificação ou geração de dados (arquivo JS)
4. Envio de dados à GPU (arquivo JS)
5. Renderizar dados (arquivo JS)

Square.html

```
<!DOCTYPE html>
<html>
<head>
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main()
{
    gl_Position = vPosition;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main()
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

Shaders

- Escolhemos nomes aos shaders para que possam ser utilizados no arquivo JS
- Estes são shaders triviais, que simplesmente atribuem valores para duas variáveis cruciais:
  - gl_Position
  - gl_FragColor
- Note que ambos shaders são programas completos
- Necessário escolher a precisão no shader de fragmentos

Square.html (cont.)

```
<script type="text/javascript" src="../Common/webgl-
utils.js"></script>
<script type="text/javascript" src="../Common/
initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></
script>
<script type="text/javascript" src="square.js"></script>
</head>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas
element
</canvas>
</body>
</html>
```

square.js

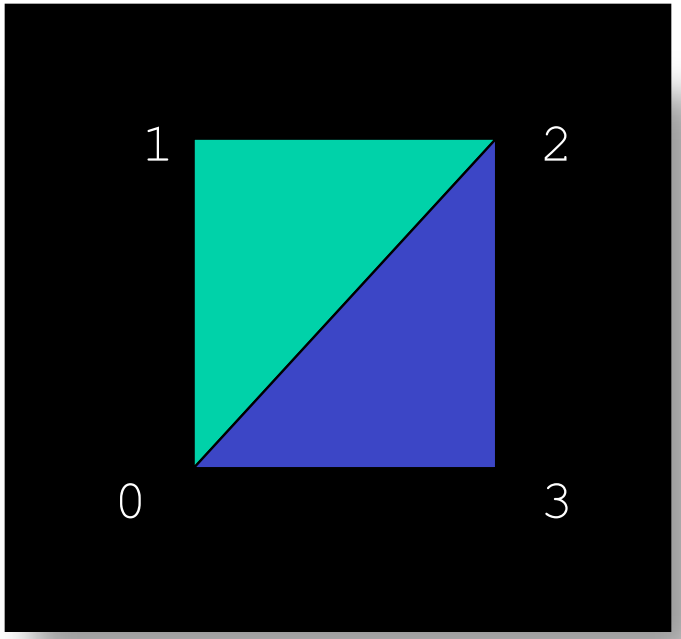
```
var gl;
var points;

window.onload = function init() {
var canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" );
    }

    // Four Vertices
    var vertices = [
        vec2( -0.5, -0.5 ),
        vec2( -0.5,  0.5 ),
        vec2(  0.5,  0.5 ),
        vec2(  0.5, -0.5)
    ];
```

square.js (cont.)

```
//  Configure WebGL
gl.viewport( 0, 0, canvas.width, canvas.height );
gl.clearColor( 0.0, 0.0, 0.0, 1.0 );

//  Load shaders and initialize attribute buffers
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

// Load the data into the GPU
var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

// Associate out shader variables with our data buffer
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
```

square.js (cont.)

```
    render();
};

function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );
}
```



Exercícios

1. Faça com que o programa agora desenhe dois triângulos, um ao lado do outro, em cores diferentes, e que um rotacione no sentido anti-horário e outro no sentido horário.
2. Agora faça com que o seu canvas ocupe a janela inteira do seu browser, garantindo essa funcionalidade mesmo após o redimensionamento.