

OPEN GRAPHICS LIBRARY (OPEN GL)

MODELAGEM E CONSTRUÇÃO DE APLICAÇÕES 3D

LABORATÓRIO

Prof. M. Sc. Will Machado

OPEN GL

**UMA ESPECIFICAÇÃO DE UMA APPLICATION
PROGRAMMING INTERFACE (API) PARA A
CRIAÇÃO DE APLICAÇÕES GRÁFICAS**

INTRODUÇÃO

- Introduzido em 1992 pela Silicon Graphics.
- Coleção de rotinas para gerar e exibir cenas 3D interativas:
 - modelos ou formas com primitivas geométricas, descrevendo modelos matemáticos de objetos;
 - pontos no espaço 2D e 3D;
 - ponto para visualização da cena;
 - cor das formas e modelos (automática ou com parâmetros de iluminação);
 - conversão dos modelos matemáticas e as cores dos objetos para pixels na tela.

BIBLIOTECA GRÁFICA

pertence a
biblioteca GL

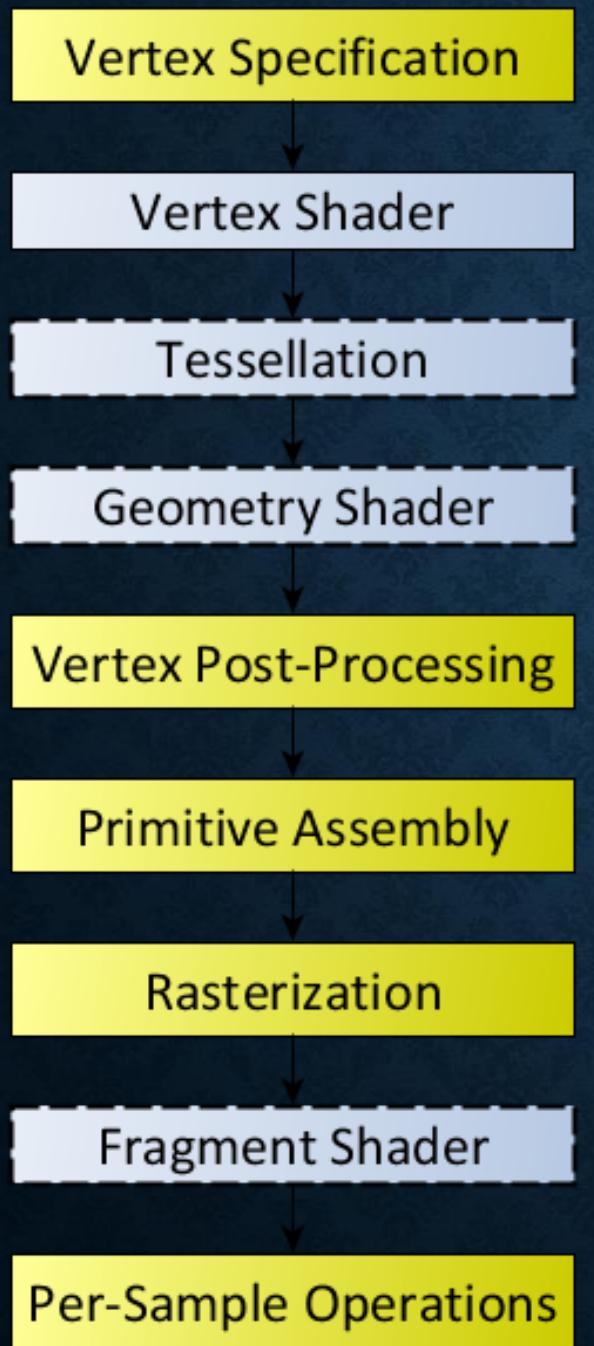
Nome da função
`glVertex3f(x, y, z)`

`x, y, z` são floats

`glVertex3fv(p)`

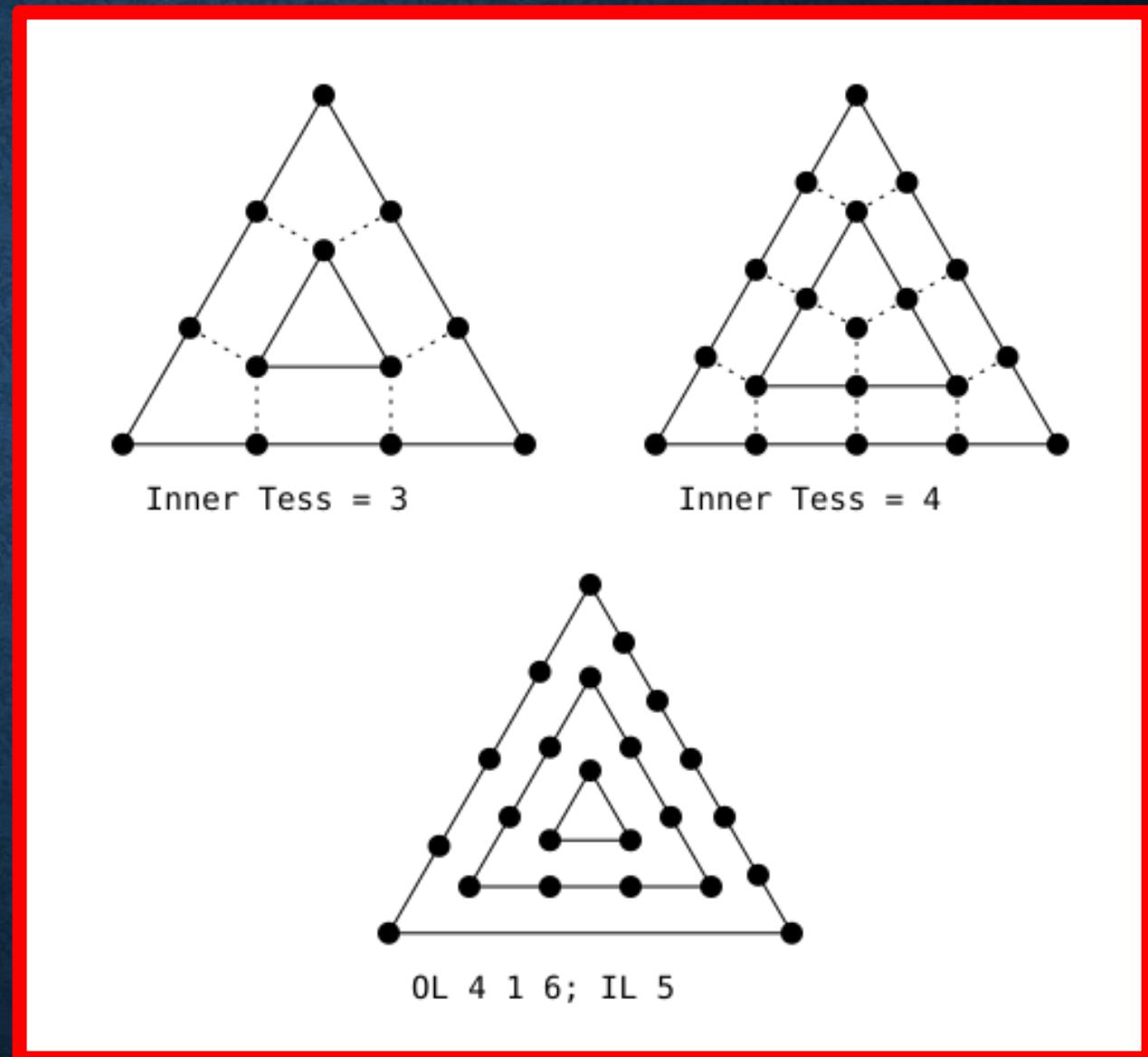
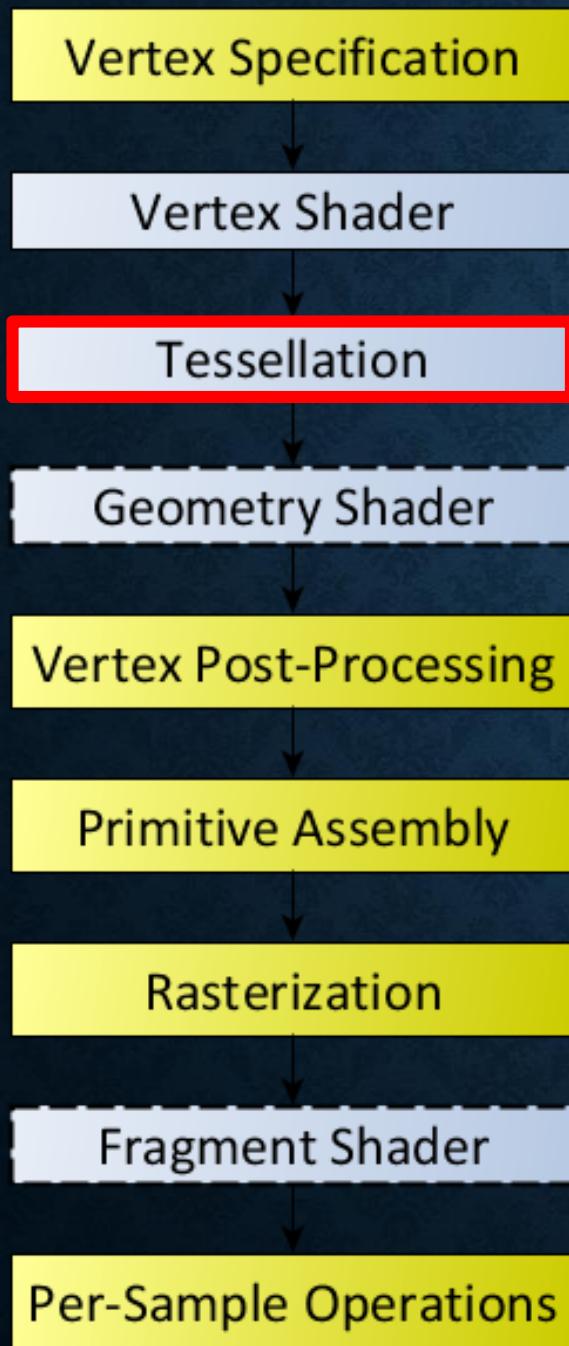
`p` é um ponteiro para um vetor

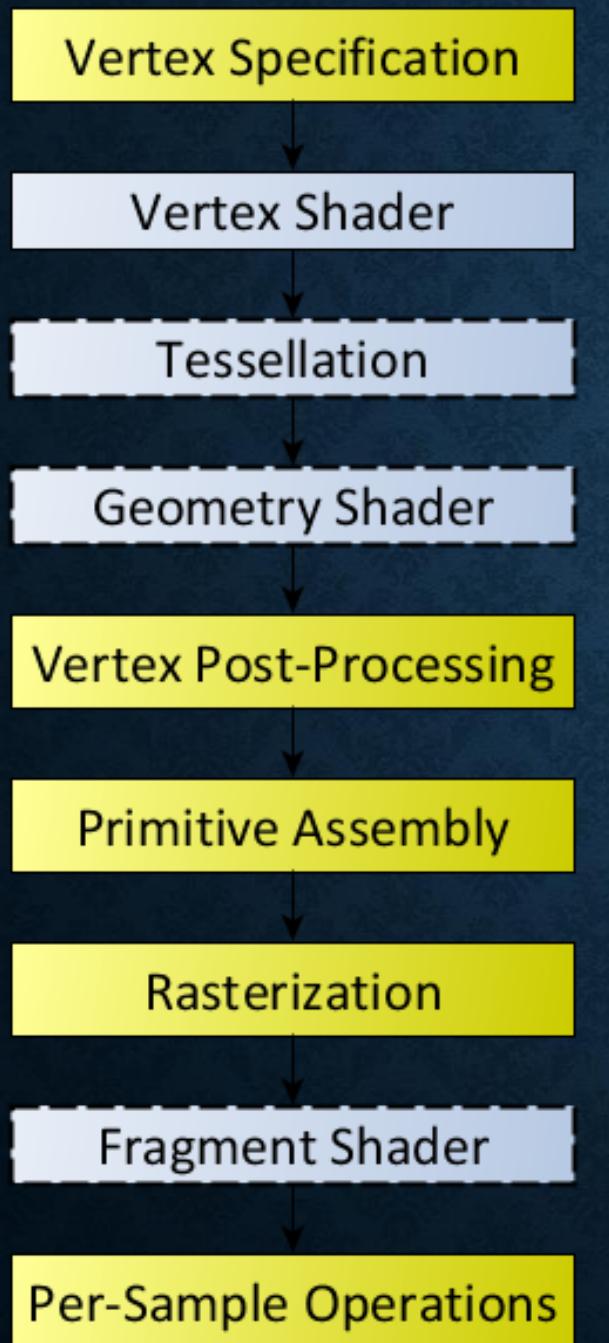
C Type	Bitdepth	Description
GLboolean	1+	A boolean value, either GL_TRUE or GL_FALSE
GLbyte	8	Signed, 2's complement binary integer
GLubyte	8	Unsigned binary integer
GLshort	16	Signed, 2's complement binary integer
GLushort	16	Unsigned binary integer
GLint	32	Signed, 2's complement binary integer
GLuint	32	Unsigned binary integer
GLfixed	32	Signed, 2's complement 16.16 integer
GLint64	64	Signed, 2's complement binary integer
GLuint64	64	Unsigned binary integer



1) Processamento de vértices :

- Controla o processamento de vértices individuais.
- Opcionais de mosaico primitivo (ex. GL_POINTS). Permite que patches de dados de vértices sejam subdivididos em primitivo menores.



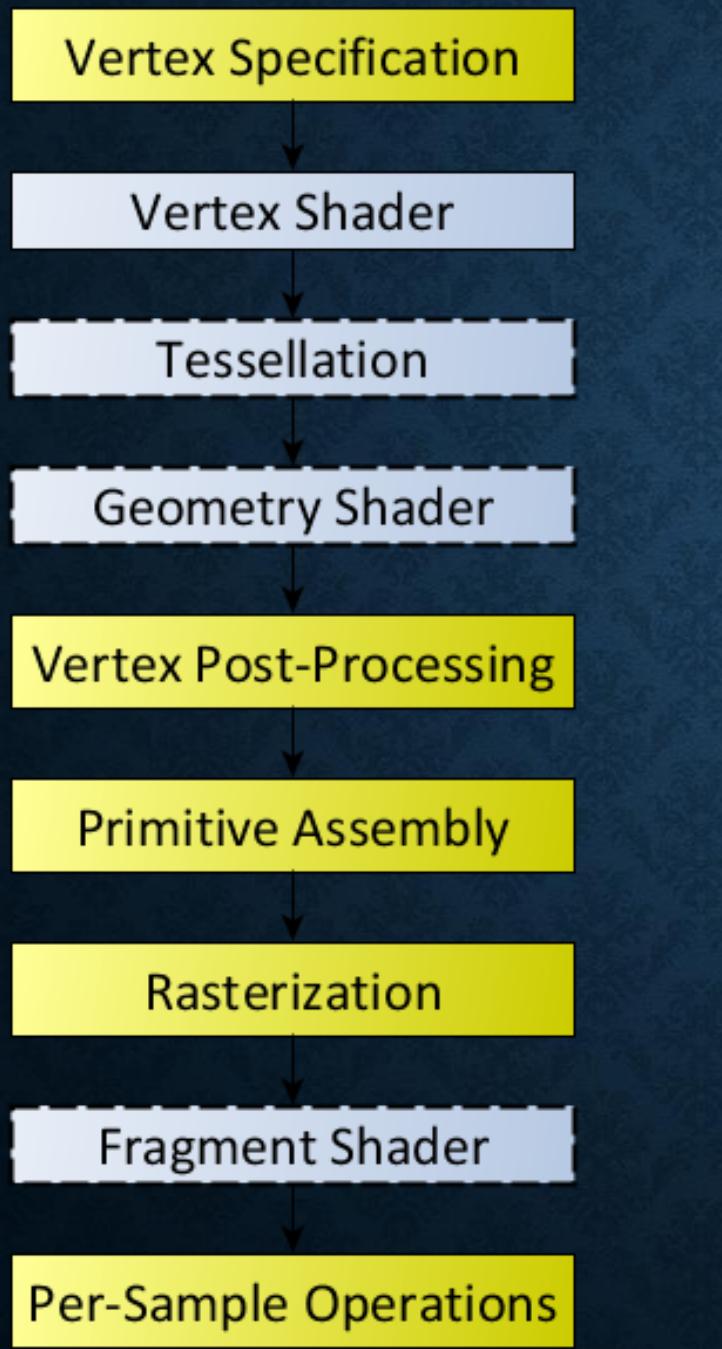


1) Processamento de vértices :

- Controla o processamento de vértices individuais.
- Opcionais de mosaico primitivo (ex. GL_POINTS). Permite que patches de dados de vértices sejam subdivididos em primitivo menores.
- Controla o processamento de primitivos (opcional).

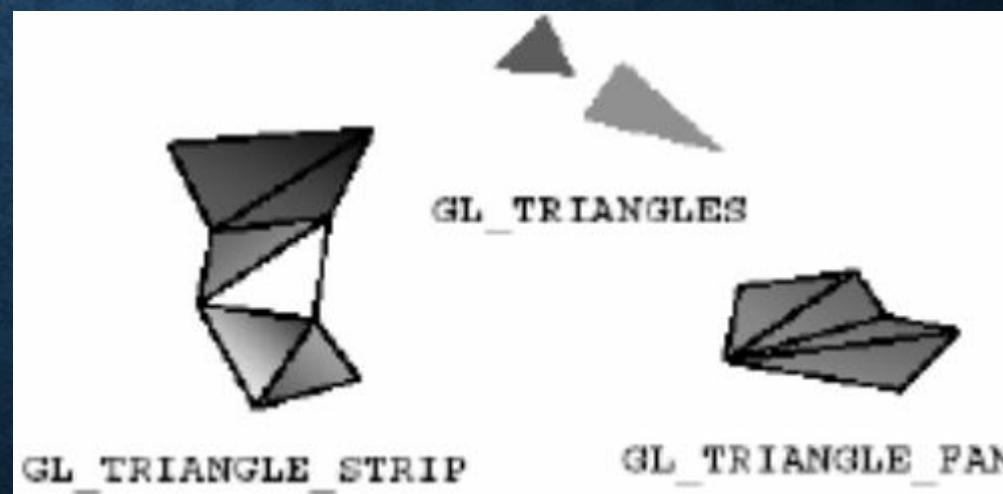
2) *Vertex Post-Processing*: as saídas do último estágio são ajustadas ou enviadas para diferentes locais.

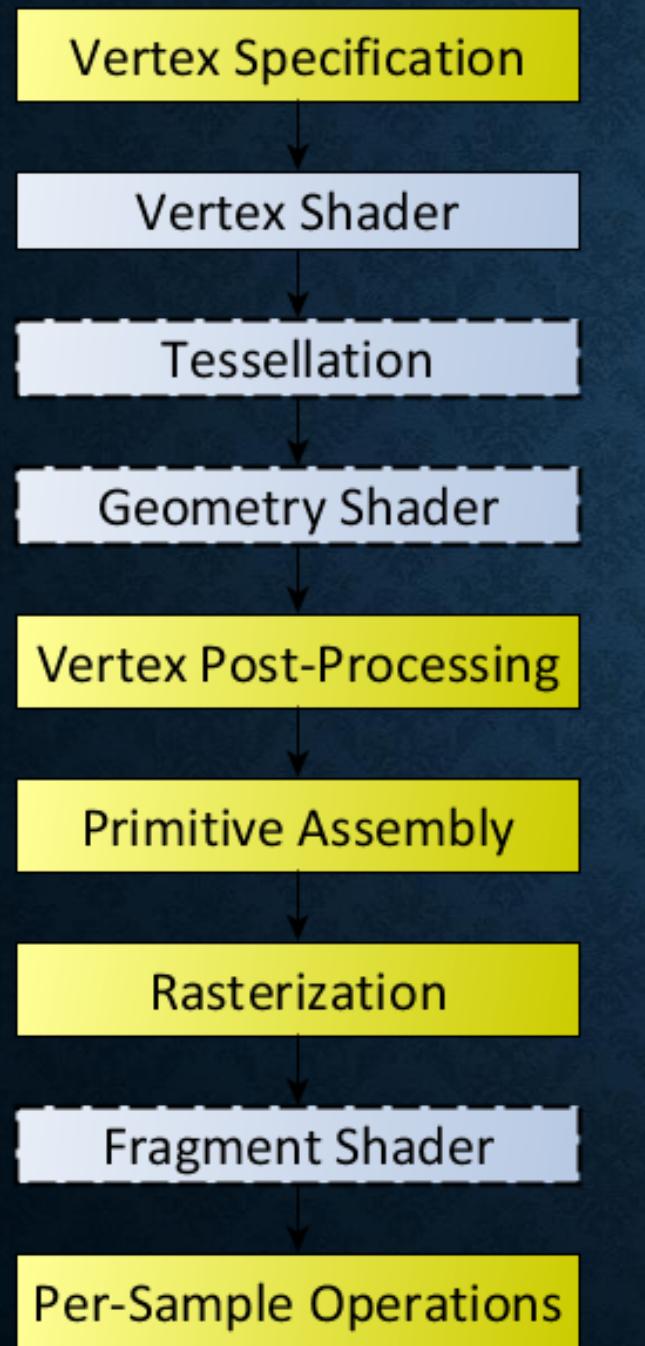
- *Transform Feedback*: registrar a saída de valores do estágio de processamento de vértices em objetos de buffer.
- *Clipping* (Recorte primitivo): Define o volume de visualização de um vértice em razão da perspectiva e da janela de visualização.



3) Processo onde os primitivos são divididos em uma sequência de primitivos de base individuais.

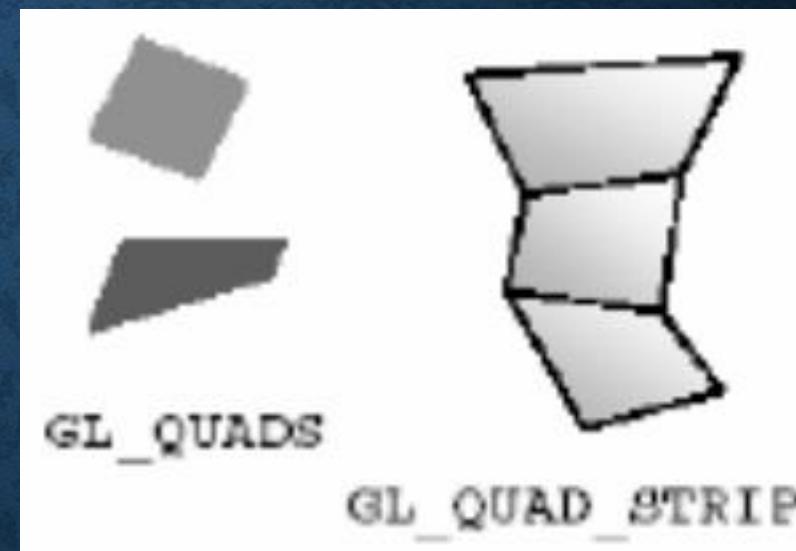
Exemplo de primitivas de linha:

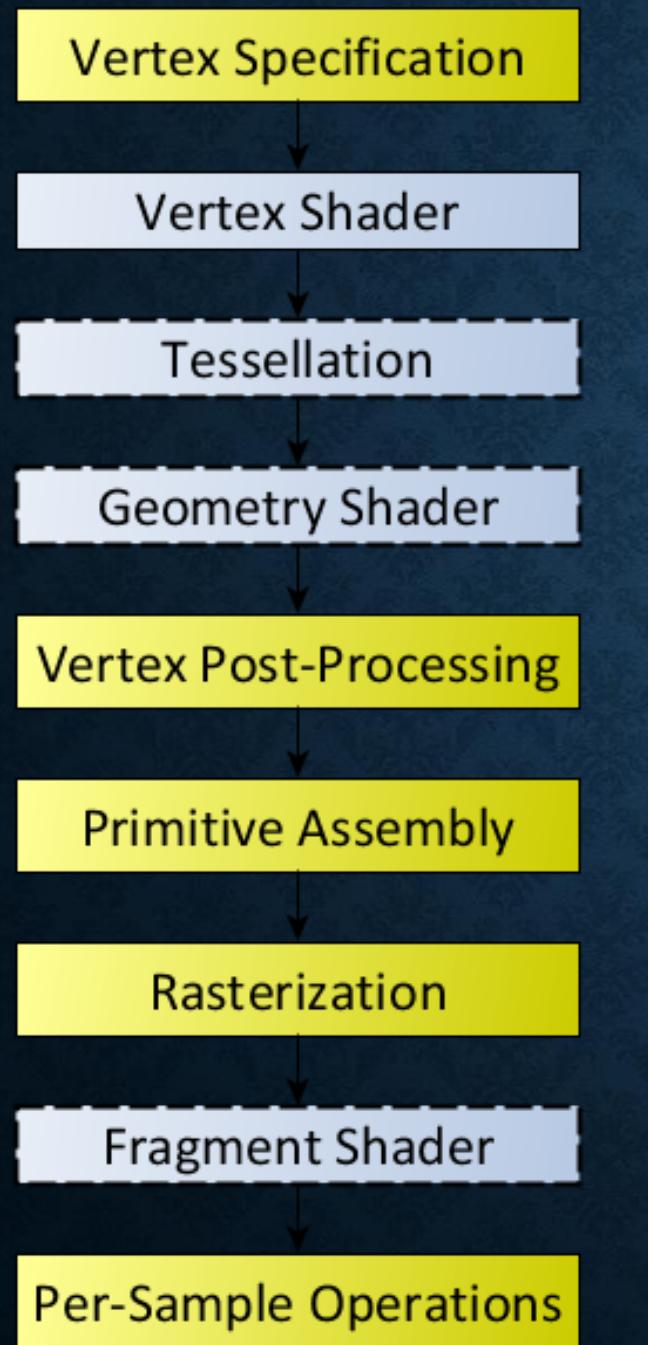




3) Processo onde os primitivos são divididos em uma sequência de primitivos de base individuais.

Exemplo de primitivas de linha:





3) Processo onde os primitivos são divididos em uma sequência de primitivos de base individuais.

4) Conversão de varredura e interpolação de parâmetro primitivo, o que gera uma série de fragmentos .

- Um Fragment Shader processa cada fragmento. Cada fragmento gera uma série de resultados.

5) Per-Sample_Processing , incluindo, mas não se limitando a:

- Teste de Tesoura
- Teste de Estêncil
- Teste de profundidade
- Misturando
- Operação Lógica
- Máscara de escrita

PER-SAMPLE OPERATIONS

- Teste de propriedade de pixel: Como o Framebuffer padrão pertence a um recurso externo ao OpenGL, é possível que pixels específicos do framebuffer padrão não sejam propriedade do OpenGL. E, portanto, o OpenGL não pode gravar nesses pixels. Fragmentos direcionados a tais pixels são, portanto, descartados neste estágio do pipeline.

De modo geral, se a janela para a qual você está renderizando estiver parcialmente obscurecida por outra janela, os pixels cobertos pela outra janela não serão mais propriedade da OpenGL e, portanto, falharão no teste de propriedade. Quaisquer fragmentos que cobrem esses pixels serão descartados. Isso também inclui operações de limpeza do *framebuffer* .

Observe que este teste afeta apenas a renderização para o framebuffer padrão. Ao renderizar para um objeto Framebuffer , todos os fragmentos passam neste teste.

PER-SAMPLE OPERATIONS

- Teste de tesoura: Uma área retangular do framebuffer de destino pode ser designada como a única área válida para renderização. Todos os fragmentos direcionados a pixels fora desse retângulo serão descartados neste estágio.
- Teste de estêncil: O teste de estêncil, quando ativado, pode fazer com que um fragmento seja descartado com base em uma operação bit a bit entre o valor do estêncil do fragmento e o valor do estêncil armazenado no Buffer de Estêncil atual na posição de amostra desse fragmento. Isso permite ao usuário estabelecer valores de estêncil em uma passagem de renderização e, em seguida, selecionar fragmentos condicionalmente com base neste padrão.

PER-SAMPLE OPERATIONS

- Atualização de consulta de oclusão: Se o fragmento passou no teste de profundidade (e apenas passando no teste de profundidade é verificado), então, neste ponto, uma consulta de oclusão ativa será considerada como tendo um fragmento aprovado. Portanto, se a consulta for uma consulta `GL_SAMPLES_PASSED` , o contador será incrementado. Se for uma das consultas booleanas, o valor booleano será definido como verdadeiro.

Observe que chegar a esse ponto não garante que qualquer valor específico seja realmente gravado em um buffer específico. As máscaras de gravação abaixo podem desativar a gravação em buffers específicos. As configurações de buffer de desenho do framebuffer podem ignorar efetivamente um ou mais valores de fragmento. Na verdade, um objeto framebuffer pode estar inteiramente vazio , sem nenhuma imagem anexada.

PER-SAMPLE OPERATIONS

- Misturando: Cada uma das cores no fragmento pode ser combinada com a cor de pixel correspondente no buffer em que o fragmento será gravado. O resultado desta operação de mistura é o que será escrito.

Se a imagem que está sendo lida estiver no espaço de cor sRGB e GL_FRAMEBUFFER_SRGB estiver habilitado no momento , a cor lida para a operação de mesclagem será convertida em RGB linear antes de mesclar com a cor do fragmento.

- Operação lógica: As cores dos fragmentos podem ser combinadas condicionalmente com o valor correspondente no framebuffer, executando operações booleanas neles. Isso substitui a mesclagem e só funciona para cores que estão gravando em imagens inteiras (normalizadas ou não). Formatos de imagem que não usam gravação sRGB em imagens sRGB.

PER-SAMPLE OPERATIONS

- Máscara de escrita: As gravações em buffers específicos podem ser mascaradas. Isso também permite mascarar componentes específicos dessas gravações, de forma que você só possa atualizar o componente de cor vermelha se desejar. Cada buffer de saída pode ter sua própria máscara.

PER-SAMPLE OPERATIONS

- **Teste de profundidade:** O teste de profundidade, quando ativado, permite que um fragmento seja selecionado com base em um teste condicional entre o valor de profundidade do fragmento e o valor de profundidade armazenado no Depth Buffer atual na posição de amostra desse fragmento. Isso é útil para fazer com que a geometria fique oculta atrás de outra geometria. A geometria mais próxima estabelece valores de profundidade que mascaram a renderização de quaisquer fragmentos atrás deles, usando a condição de teste de profundidade adequada.

GL_POINTS

```
glBegin(GL_POINTS);

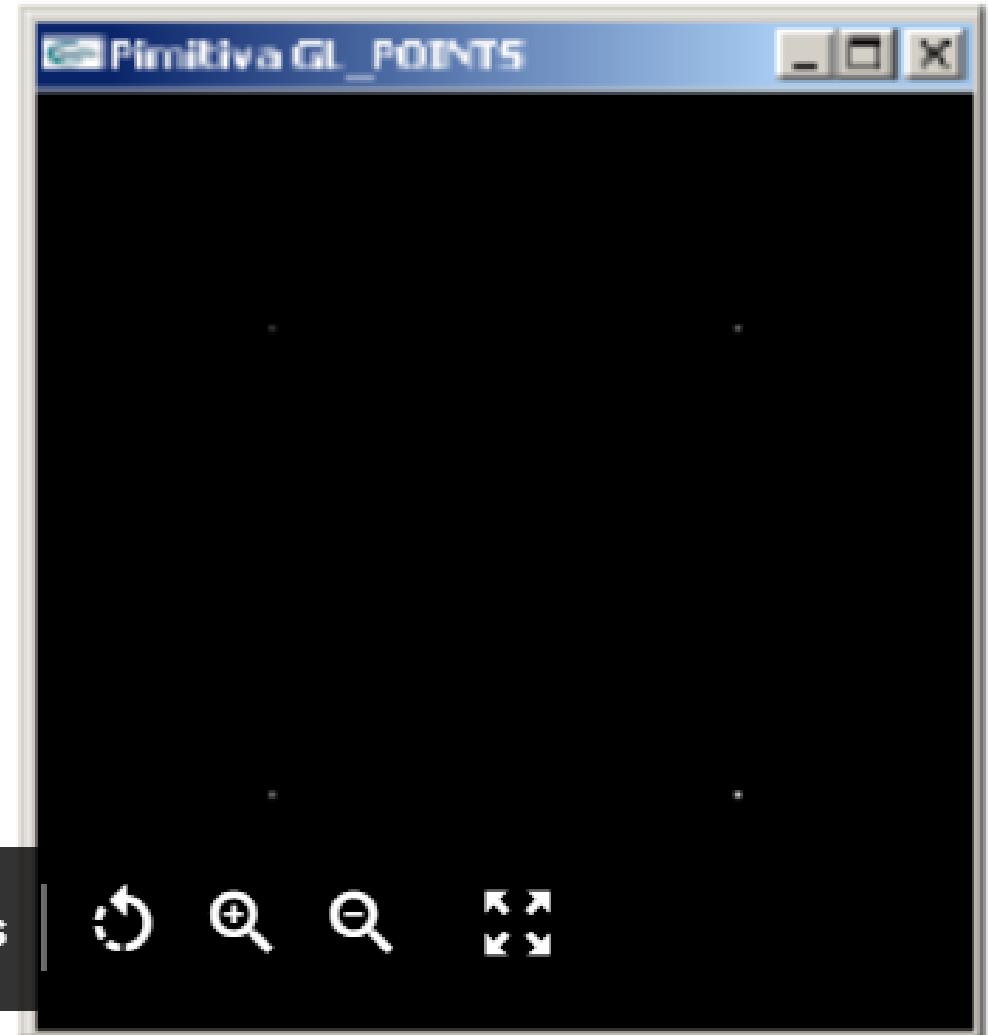
    glVertex3f (0.25, 0.25, 0.0);

    glVertex3f (0.75, 0.25, 0.0);

    glVertex3f (0.75, 0.75, 0.0);

    glVertex3f (0.25, 0.75, 0.0);

glEnd();
```



GL_LINE_STRIP

```
glBegin(GL_LINE_STRIP);

    glVertex3f (0.25, 0.25, 0.0);

    glVertex3f (0.75, 0.75, 0.0);

    glVertex3f (0.30, 0.45, 0.0);

glEnd();
```



GL_LINE_LOOP

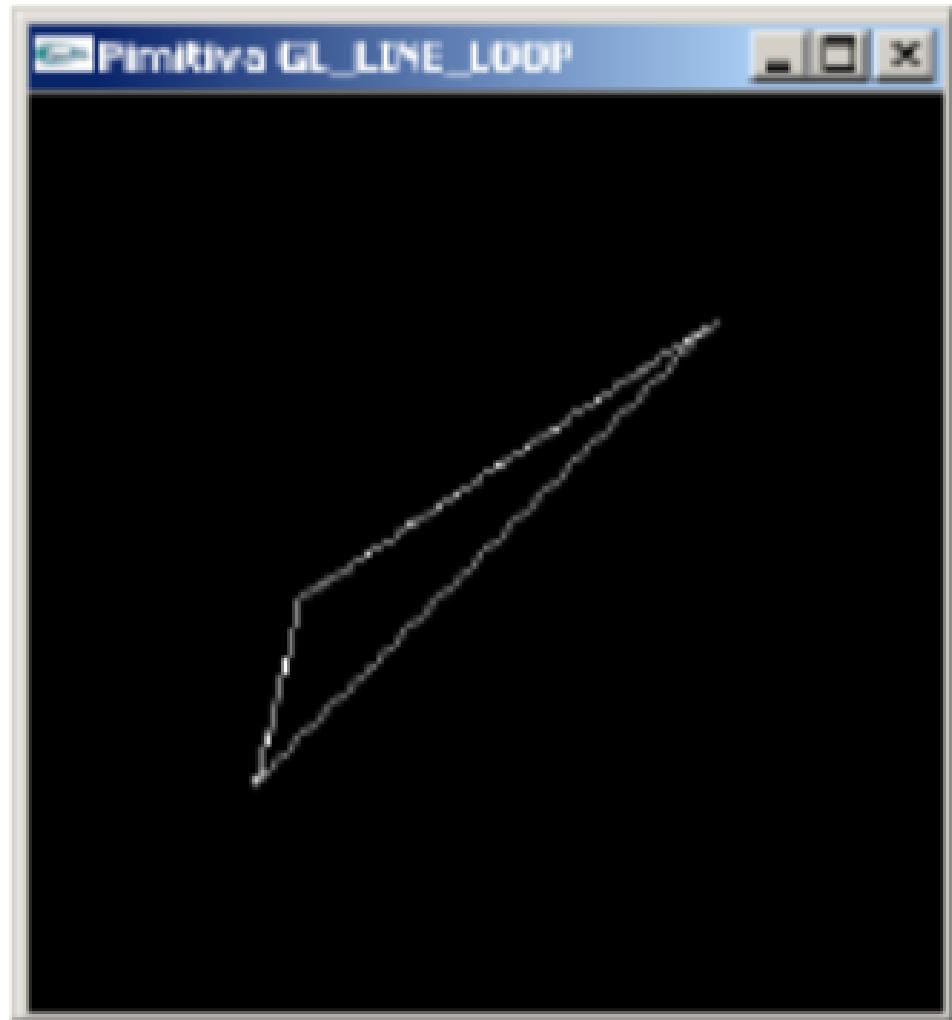
```
glBegin(GL_LINE_LOOP);

    glVertex3f (0.25, 0.25, 0.0);

    glVertex3f (0.75, 0.75, 0.0);

    glVertex3f (0.30, 0.45, 0.0);

glEnd();
```



Bibliotecas



Socially Aware
Organisations
and Technologies
Impact and Challenges

17th IFIP WG 8.1 International Conference
on Informatics and Semantics in Organisations, ICISO 2016,
Campinas, Brazil, August 1–3, 2016, Proceedings

Springer

- **GLU - OpenGL Utility Library:** contém várias rotinas que utilizam os comandos OpenGL de baixo nível para executar tarefas como, por exemplo, definir as matrizes para projeção e orientação da visualização, e fazer o rendering de uma superfície. Esta biblioteca é fornecida como parte de cada implementação de OpenGL, e suas funções usam o prefixo glu [Woo 1999].
- **GLUT - OpenGL Utility Toolkit:** é um toolkit independente de plataforma, que inclui alguns elementos GUI (Graphical User Interface), tais como menus pop-up e suporte para joystick. Esta biblioteca, escrita por Mark Kilgard, não é domínio público, mas é free. O seu principal objetivo é esconder a complexidade das APIs dos diferentes sistemas de janelas. As funções desta biblioteca usam o prefixo glut. É interessante comentar que a GLUT substitui a GLAUX, uma biblioteca auxiliar OpenGL que havia sido criada para facilitar o aprendizado e a elaboração de programas OpenGL independente do ambiente de programação (Linux, Windows, etc.) [Woo 1999, Wright 2000].

- **GLX - OpenGL Extension to the X Window System:** fornecido como um "anexo" de OpenGL para máquinas que usam o X Window System. Funções GLX usam o prefixo glX. Para Microsoft Windows 95/98/NT, as funções WGL fornecem as janelas para a interface OpenGL. Todas as funções WGL usam o prefixo wgl. Para IBM/OS2, a PGL é a Presentation Manager para a interface OpenGL, e suas funções usam o prefixo pgl. Para Apple, a AGL é a interface para sistemas que suportam OpenGL, e as funções AGL usam o prefixo agl [Woo 1999].
- **FSG - Fahrenheit Scene Graph:** é um toolkit orientado à objetos e baseado em OpenGL, que fornece objetos e métodos para a criação de aplicações gráficas 3D interativas. FSG, que foi escrito em C++ e é separado de OpenGL, fornece componentes de alto nível para criação e edição de cenas 3D, e a habilidade de trocar dados em outros formatos gráficos [Woo 1999].

- **SDL - Simple DirectMedia Layer:** biblioteca de desenvolvimento de software de plataforma cruzada projetada para fornecer uma camada de abstração de hardware para componentes de hardware de multimídia de computador . Os desenvolvedores de software podem usá-lo para escrever jogos de computador de alto desempenho e outros aplicativos de multimídia que podem ser executados em muitos sistemas operacionais , como Android , iOS , Linux , macOS e Windows .

SDL gerencia vídeo , áudio , dispositivos de entrada , CD-ROM , threads , carregamento de objeto compartilhado , rede e temporizadores. Para gráficos 3D, ele pode lidar com um contexto OpenGL , Vulkan ou Direct3D11 (o direct3d versão 9 mais antigo também é compatível).

- **GLFW - Graphics Library Framework:** biblioteca pequena escrita em linguagem C que permite a criação e gerenciamento de janelas em um contexto OpenGL, tornando possível também o uso de múltiplos monitores e modos de vídeo. Fornece acesso de leitura de teclado, mouse e joysticks. A API fornece uma fina camada de abstração multiplataforma, principalmente para aplicações cuja única saída gráfica é através da API OpenGL. Embora a GLFW seja muito útil para desenvolver aplicações OpenGL multiplataforma, desenvolvedores voltados a apenas uma plataforma também podem se beneficiar ao evitar terem que lidar com APIs mal feitas específicas da plataforma.

Um motivo para que bibliotecas como a GLFW sejam necessárias é que a OpenGL por si só não fornece mecanismos para criar o contexto necessário e gerenciar janelas, entradas do usuário, tempo, etc. Existem muitas outras bibliotecas disponíveis para auxiliar no desenvolvimento OpenGL.

- **GLFW - Graphics Library Framework:**

As mais comuns são freeglut (uma implementação de código aberto da GLUT) e SDL.

No entanto, a **freeglut** é focada em ser um clone estável da GLUT, enquanto a SDL é muito grande para os interesses de algumas pessoas e nunca teve OpenGL como seu principal foco. A GLFW está baseada na suposição de que há espaço para uma biblioteca leve e moderna para gerenciar contextos OpenGL, janelas e dispositivos de entrada.

ESTRUTURA BÁSICA GLUT

```
1 #include<GL/freeglut.h>
2 #include<iostream>
3
4 void desenha(void) {
5     glClear( GL_COLOR_BUFFER_BIT );
6     glFlush();
7 }
8
9 int main(int argc, char* argv[])
10 {
11     glutInit(&argc, argv);
12     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
13     glutInitWindowSize(800, 600);
14     glutInitWindowPosition(300, 100);
15     glutCreateWindow("Ola Glut");
16     glutDisplayFunc(desenha);
17     glColor3f( 0, 0, 1, 0 );
18     glutMainLoop();
19     return 0;
20 }
```

```
#include<GL/freeglut.h>
#include<iostream>

void desenha(void) {
    glClear( GL_COLOR_BUFFER_BIT );
    glFlush();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(800,600);
    glutCreateWindow("Ola Glut");
    glutDisplayFunc(desenha);
    glClearColor( 0, 0, 1, 0 );
    glutMainLoop();
    return 0;
}
```

```
4  void desenha(void) {  
5      glClear( GL_COLOR_BUFFER_BIT );  
6  
7      gluOrtho2D(-3, 3, -3, 3);  
8  
9      glBegin (GL_LINES);  
10     glVertex2f (-2, 0);  
11     glVertex2f (2, 0);  
12     glEnd();  
13  
14     glFlush();  
15 }
```



REDIMENSIONAMENTO

```
4  void desenha(void) {
5      glClear( GL_COLOR_BUFFER_BIT );
6
7      glMatrixMode(GL_PROJECTION);
8      glLoadIdentity();
9      gluOrtho2D(-6, 6, -6, 6);
10
11     glMatrixMode(GL_MODELVIEW);
12     glLoadIdentity();
13     //glBegin(GL_LINES);
14     glBegin(GL_TRIANGLES);
15         glColor3f(0,1,0);
16         glVertex2f(-2,0);
17         glVertex2f(2,0);
18         glColor3f(1,0,0);
19         glVertex2f(0,2);
20     glEnd();
21
22     glFlush();
23 }
```

IMPLEMENTAÇÕES 2D

- Pontos
- Linhas
- Triângulos
- Quadrado
- Polígonos

PONTOS

```
glBegin(GL_POINTS);  
    glColor3f(0.0f, 0.0f, 0.0f);  
    glVertex2i(100, 50);  
    glVertex2i(100, 130);  
    glVertex2i(150, 130);  
glEnd();
```



INTERAÇÃO VIA TECLADO

```
6  void desenha(void) {
7      glClear( GL_COLOR_BUFFER_BIT );
8
9      glMatrixMode(GL_PROJECTION);
10     glLoadIdentity();
11     gluOrtho2D(-6, 6, -6, 6);
12
13     glScalef(escala, escala, 0);
14
15     glMatrixMode(GL_MODELVIEW);
16     glLoadIdentity();
17     //glBegin(GL_LINES);
18     glBegin(GL_TRIANGLES);
19         glColor3f(0,1,0);
20         glVertex2f(-2,0);
21         glVertex2f(2,0);
22         glColor3f(1,0,0);
23         glVertex2f(0,2);
24     glEnd();
25
26     glFlush();
27 }
```



```
1  #include<GL/glut.h>
2  #include<iostream>
3
4  GLfloat escala = 1;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29  void listeningKey (unsigned char tecla,
30                      GLint x, GLint y) {
31
32     switch(tecla) {
33         case '+':   escala++;
34                     break;
35         case '-':   escala--;
36                     break;
37     }
38     desenha();
39
40
41
42
43
44
45     glutInitWindowPosition(300,100);
46     glutCreateWindow("Ola Glut");
47     glutKeyboardFunc(listeningKey);
48     glutDisplayFunc(desenha);
```



INTERAÇÃO VIA MOUSE

```
1 #include<GL/glut.h>
2 #include<stdlib.h>
3 #include<stdio.h>
4 #include<time.h>
5
6 float r,g,b,x,y;
7 bool check=true;
8
9 void mouse(int button, int state, int mousex, int mousey)
10 {
11     if(button==GLUT_LEFT_BUTTON)
12     //if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
13     {
14         check=true;
15         x = mousex;
16         y = 480-mousey;
17         r=(rand()%10)/10.0;
18         g=(rand()%10)/10.0;
19         b=(rand()%10)/10.0;
20     }
21     else if(button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
22     {
23         glClearColor(1, 1, 1, 0);
24         glClear(GL_COLOR_BUFFER_BIT);
25         check = false;
26     }
27     glutPostRedisplay();
28 }
```

```
30 void display(void)
31 {
32     glColor3f(r,g,b);
33     glPointSize(50);
34     glMatrixMode(GL_PROJECTION);
35     glLoadIdentity();
36     gluOrtho2D(0.0, 640.0, 0.0, 480.0);
37     if(check)
38     {
39         glBegin(GL_POINTS);
40             glVertex2i(x,y);
41         glEnd();
42     }
43     glFlush();
44 }
```

```
46 int main(int argc, char** argv)
47 {
48     srand(time(NULL));
49     glutInit(&argc, argv);
50     glutInitWindowSize(640,480);
51     glutInitWindowPosition(10,10);
52     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
53     glutCreateWindow("Interacao via Mouse");
54     glClearColor(1, 1, 1, 0);
55     glClear(GL_COLOR_BUFFER_BIT);
56     glutDisplayFunc(display);
57     glutMouseFunc(mouse);
58     glutMainLoop();
59 }
```



ANIMAÇÃO