



Introdução ao OpenGL

Parte II

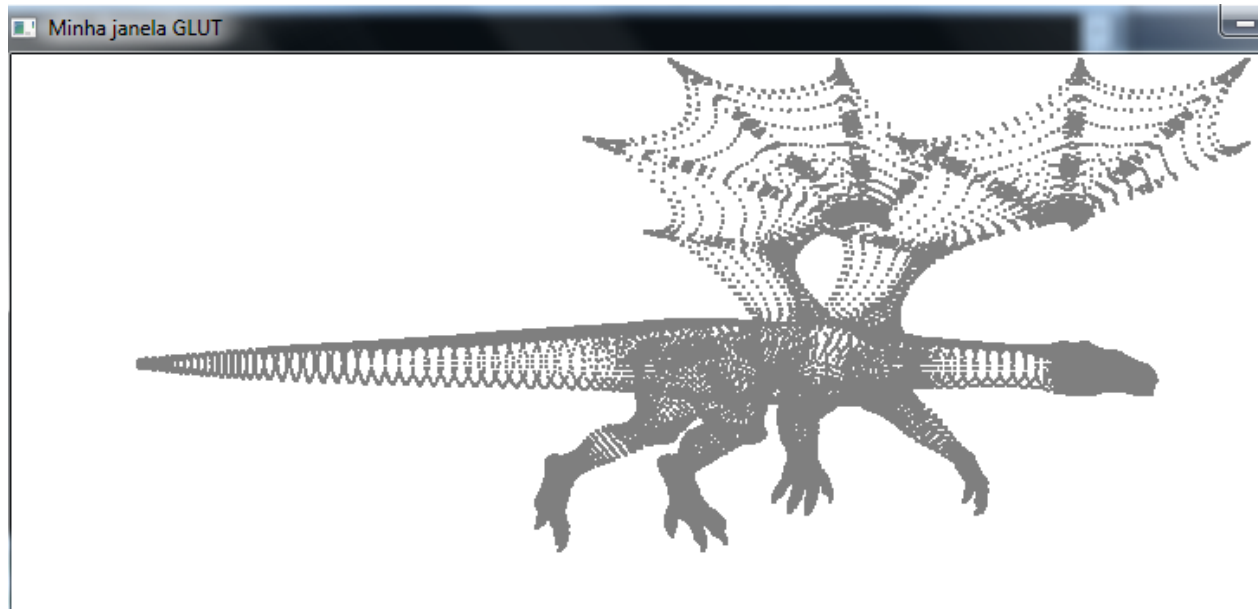
Gilda Aparecida de Assis

gilda1@gmail.com

Agradecimentos ao professor Eduardo Filgueiras Damasceno por disponibilizar o material para construção da disciplina

Como carregar objetos 3D de arquivos?

- Especificar modelos 3D através de **glVertex3f** em OpenGL para modelos complexos é uma tarefa inviável.
- Personagens e cenário são criados em programas de modelagem 3D como 3D Studio Max, Blender, etc
- Os modelos podem ser exportados do 3D Studio Max, Blender ou outros programas de modelagem 3D para o formato **OBJ**



O que é um arquivo OBJ?

- O formato **OBJ** carrega informações sobre a malha poligonal, mapeamento de texturas e materiais.
- O arquivo é composto por vértices (linhas que começam com “**v**”), normais (linhas com “**vn**”), mapeamentos de texturas (linhas com “**vt**”) e faces (linhas que começam com “**f**”).
- É possível ainda que o arquivo seja formado por grupos.
 - Os dados de um grupo são delimitados por uma linha com instrução: “**g** <nome do grupo>” até encontrar um novo “**g**”.

O que é um arquivo OBJ?

• Formato para leitura das faces:

- Somente vértices:
 - `v1 v2 v3`
- Completo:
 - `v1/t1/n1 v2/t2/n2 v3/t3/n3`
- Vértices e normais:
 - `v1//n1 v2//n2 v3//n3`
- Vértices e texturas:
 - `v1/t1 v2/t2 v3/t3`

O que é um arquivo OBJ?

- As faces não apontam diretamente para os vértices, mas sim para o seu índice no vetor de vértices do arquivo.
- Os índices começam por 1 no arquivo.
- As faces podem ser triângulos, quadriláteros ou polígonos quaisquer; portanto, não é possível prever a quantidade de vértices que terá antes de ler o arquivo.
- Linhas que começam com “#” são comentários e devem ser ignoradas.



Um exemplo de OBJ- Parte 1

Blender3D v249 OBJ File:
untitled.blend

www.blender3d.org

mtllib cube.mtl

v 1.000000 -1.000000 -1.000000

v 1.000000 -1.000000 1.000000

v -1.000000 -1.000000 1.000000

v -1.000000 -1.000000 -1.000000

v 1.000000 1.000000 -1.000000

v 0.999999 1.000000 1.000001

v -1.000000 1.000000 1.000000

v -1.000000 1.000000 -1.000000

vt 0.748573 0.750412

vt 0.749279 0.501284

vt 0.999110 0.501077

vt 0.999455 0.750380

vt 0.250471 0.500702

vt 0.249682 0.749677

vt 0.001085 0.750380

vt 0.001517 0.499994

vt 0.499422 0.500239

vt 0.500149 0.750166

vt 0.748355 0.998230

vt 0.500193 0.998728

vt 0.498993 0.250415

vt 0.748953 0.250920

vn 0.000000 0.000000 -1.000000

vn -1.000000 -0.000000 -0.000000

vn -0.000000 -0.000000 1.000000

Um exemplo de OBJ – Parte 2

vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
usemtl Material_ray.png
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5

f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8

Como carregar um OBJ?

- Forma mais simples: Nuvem de Pontos

- Criar uma função loadOBJ para ler o arquivo, gravar os dados em uma lista de vértices, e retornar false se algo deu errado.
- `Id_objeto= glGenLists(qtde_listas)`: Aloca um intervalo contínuo de índices. Se *qtde_listas* for igual a 1, gera um único índice que é retornado pela função.
- `void glNewList (Id_objeto, modo)`: Especifica o início do display list. Todos comandos OpenGL antes do `glEndList()` são guardados no display list.
- *Id_objeto*: valor retornado por `glGenLists(1)`.
- Modo: `GL_COMPILE_AND_EXECUTE` ou `GL_COMPILE` (indica que os comandos não devem ser executados à medida que são inseridos no display list).

Como carregar um OBJ?

- Forma mais simples: Nuvem de Pontos
 - void `glEndList` (): indica o fim do display list.
 - void `glCallList` (*id_objeto*): Executa o display list referenciado por *id_objeto*. Os comandos são executados na ordem que foram especificados.
- O uso de display lists reduz o número de chamada de funções e também o processamento para geração dos objetos, pois os diversos cálculos necessários são realizados uma única vez e apenas comandos OpenGL são armazenados na display list para processamento futuro.

Função para Carregar OBJ – Parte 1

GLuint obj;

void loadPointsObj(char *fname)

{

FILE *fp;

int read;

GLfloat x, y, z;

char ch;

int resp;

obj = glGenLists(1);



fp = fopen(fname, "r");

if (fp==NULL)

{

printf("Não foi possível abrir o arquivo %s\n", fname);



scanf("%i", &resp);

exit(1);

}

glPointSize(2.0);

Função para Carregar OBJ – Parte 1

```
glNewList(obj, GL_COMPILE);   
{  
    glPushMatrix();  
    glBegin(GL_POINTS);  
    while (!(feof(fp)))  
    {  
        read = fscanf(fp, "%c %f %f %f", &ch, &x, &y, &z);  
        if (read == 4 && ch == 'v')  
        {  
            glVertex3f(x, y, z);  
        }  
    }  
    glEnd();  
}  
glPopMatrix();  
glEndList();   
fclose(fp);  
}
```


Função para Desenhar OBJ

```
void drawObj()  
{  
    glPushMatrix();  
    glColor3f(0.5, 0.5, 0.5);  
    glRotatef(60, 0.0, 1.0, 0.0);  
    glCallList(obj);  
    glPopMatrix();  
}  
  
void renderScene(void)  
{  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
    glClearColor(1.0, 1.0, 1.0, 0.0);  
    glLoadIdentity();  
    drawObj();  
    glutSwapBuffers();  
}
```

Função Main OBJ

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Minha janela GLUT");
    printf("Testando Opengl...");
    loadPointsObj("Dargon.obj");
    glutDisplayFunc(renderScene);
    glutMainLoop();

    return 0;
}
```



Arquivo obj está na
pasta do projeto

Como carregar um OBJ?

- Problemas com nuvem de pontos
 - Para renderizar o modelo e aplicar textura é necessário utilizar faces como triângulos, polígonos, etc. Para isso, é necessário criar uma função `loadFacesObj(char *fname);`
 - O obj não contém cabeçalhos contendo a quantidade de vértices e faces. Não há nenhuma maneira fácil de determinar quantas entradas em cada array, somente contando...