PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Projeto 1
Computação Gráfica - INF1761

Alunos:
Henrique Soares - 1710466
Marcos Vinicius Araujo - 1910869
Felipe Gonzalez - 1910438

Professor:
Waldemar Celes

1. Criação de novas formas (Autor: Felipe Gonzalez)

Para criar uma cena mais diversa, decidimos instanciar uma pirâmide. Para isso, tivemos que criar o arquivo pyramid_array.cpp, onde definimos os vetores de coordenadas, normais e índices dos pontos da pirâmide. Para isso, o arquivo cube_array.cpp como referência. Todos os pontos foram definidos nos vetores de índices e de coordenadas seguindo a ordem anti-horária. Os vetores normais foram calculados manualmente.

```
m_normals{
  // back face: counter clockwise
   0.0f, 0.8320502943378437f,-0.5547001962252291f,
   0.0f, 0.8320502943378437f,-0.5547001962252291f,
   0.0f, 0.8320502943378437f,-0.5547001962252291f,
  // front face: counter clockwise
   0.0f, 0.8320502943378437f, 0.5547001962252291f,
   0.0f, 0.8320502943378437f, 0.5547001962252291f,
   0.0f, 0.8320502943378437f, 0.5547001962252291f,
  // left face: counter clockwise
  -0.5547001962252291f, 0.8320502943378437f, 0.0f,
  -0.5547001962252291f, 0.8320502943378437f, 0.0f,
  -0.5547001962252291f, 0.8320502943378437f, 0.0f,
  // right face: counter clockwise
   0.5547001962252291f, 0.8320502943378437f, 0.0f,
   0.5547001962252291f, 0.8320502943378437f, 0.0f,
   0.5547001962252291f, 0.8320502943378437f, 0.0f,
  // bottom face: counter clockwise
   0.0f,-1.0f, 0.0f,
   0.0f,-1.0f, 0.0f,
   0.0f,-1.0f, 0.0f,
   0.0f,-1.0f, 0.0f,

},
m_index{
  0,1,2,
  3,4,5,
  6,7,8,
  9,10,11,
  12,13,14,12,14,15
}
```

```
PyramidArray::PyramidArray ()
: m_coords{
  // back face: counter clockwise
  -0.5f, 0.0f, -0.5f,
  0.0f, 1.0f, 0.0f,
  0.5f, 0.0f, -0.5f,
  // front face: counter clockwise
  -0.5f, 0.0f, 0.5f,
  0.5f, 0.0f, 0.5f,
  0.0f, 1.0f, 0.0f,
  // left face: counter clockwise
  -0.5f, 0.0f, -0.5f,
  -0.5f, 0.0f, 0.5f,
  0.0f, 1.0f, 0.0f,
  // right face: counter clockwise
  0.5f, 0.0f, -0.5f,
  0.0f, 1.0f, 0.0f,
  0.5f, 0.0f, 0.5f,
  // bottom face: counter clockwise
  -0.5, 0.0f, -0.5f,
  0.5f, 0.0f, -0.5f,
  0.5f, 0.0f, 0.5f,
  -0.5f, 0.0f, 0.5f,

  },
```

2. **Fonte de luz de Spot (Autor: Marcos Vinicius Araujo Almeida)**

Para a criação da fonte de luz de spot, foi seguido o mesmo exemplo da tarefa da semana 4. Criamos uma subclasse, derivada de objtlight, que possui uma direção, uma concentração e um ângulo de abertura:

- spotlight.h:

```h
#include <memory>

class Spotlight;
typedef std::shared_ptr<Spotlight> SpotlightPtr;

#ifndef SPOTLIGHT_H
#define SPOTLIGHT_H

#include <GL/gl.h>
#include "objlight.h"

class Spotlight : public ObjLight {
  float m_pos[4];
  GLfloat m_direct[3];
  GLfloat angulo;
  GLfloat concentracao;
protected:
  Spotlight (float x, float y, float z, float w, GLfloat xD, GLfloat yD, GLfloat zD, GLfloat angle, GLfloat concentracao);
public:
  static SpotlightPtr Make (float x, float y, float z, float w, GLfloat xD, GLfloat yD, GLfloat zD, GLfloat angle, GLfloat concentracao
  ~Spotlight ();
  void SetPosition (float x, float y, float z, float w=1.0f);
  void SetDirect(GLfloat xD, GLfloat yD, GLfloat zD);
  virtual int Setup () const;
};

#endif
```

Seguimos o mesmo exemplo do construtor do objlight, criando um vetor para SpotlightPtr.

- spotlight.cpp

```cpp
Spotlight::Spotlight (float x, float y, float z, float w, GLfloat xD,
GLfloat yD, GLfloat zD, GLfloat angle, GLfloat concentracao)
: m_pos{x,y,z,w}, m_direct{xD, yD, zD}, angulo(angle), concentracao(concentracao)
{
}

Spotlight::~Spotlight ()
{
}



SpotlightPtr Spotlight::Make (float x, float y, float z, float w, GLfloat xD,
  GLfloat yD, GLfloat zD, GLfloat angle, GLfloat concentracao){
    return SpotlightPtr(new Spotlight(x, y, z, w, xD, yD, zD, angle, concentracao));
}
```

```cpp
int Spotlight::Setup () const
{
  Light::Setup();
  if (GetReference()) {
    float mat[16];
    GetReference()->GetModelMatrix(mat);
    glPushMatrix();
    glMultMatrixf(mat);
  }
  glLightfv(GL_LIGHT0+GetId(),GL_POSITION,m_pos);
  glLightfv(GL_LIGHT0+GetId(),GL_SPOT_DIRECTION, m_direct);
  glLightf(GL_LIGHT0+GetId(),GL_SPOT_CUTOFF, angulo);
  glLightf(GL_LIGHT0+GetId(),GL_SPOT_EXPONENT, concentracao);
  if (GetReference()) {
    glPopMatrix();
  }
  return 1;
}
```

Para a inicialização, retornamos um SpotlightPtr, passando como parâmetro um novo objeto do tipo Spotlight.

No método Setup, é necessário chamarmos o Setup de light e chamar as funções glLightfv (Para valores vetoriais) e glLightf (para valores do tipo GLFloat), passando a posição da spotlight, a direção para onde está apontada, seu ângulo de abertura e sua concentração.

Na main, criamos a spotlight, passando como parâmetro a posição da lâmpada (que será mostrada mais a frente), já que ela servirá para iluminar parte da mesa e adicionamos ela no construtor da Scene.

3. Múltiplas fontes de Luz (Autor: Felipe Gonzalez)

Para a criação de múltiplas fontes de luz, aproveitamos a criação da spotlight e adicionamos ela na cena. Basta apenas alterar o construtor da Scene e fazer alguns pequenos ajustes.

```
Scene::Scene (NodePtr root, LightPtr light,LightPtr light2)
: m_root(root), m_light(light),s_light(light2), m_engines()
{
}

ScenePtr Scene::Make (NodePtr root, LightPtr light, LightPtr light2)
{
  return ScenePtr(new Scene(root,light,light2));
}
```

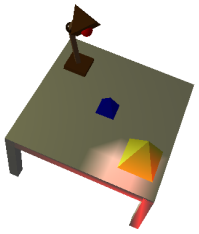
Adicionamos a nova luz no construtor da classe Scene, e logo na função de baixo, alteramos o método Make para receber a nova luz.

```
void Scene::Render (CameraPtr camera) const
{
  camera->Setup();
  if (m_light != nullptr) {
    glEnable(GL_LIGHTING);
    glEnable(GL_NORMALIZE);
    m_light->Setup();
  }
  if (s_light != nullptr) {
    glEnable(GL_LIGHTING);
    glEnable(GL_NORMALIZE);
    s_light->Setup();


  }
  m_root->Render();
}
```

No Setup, basta apenas checarmos se a luz existente na classe é null: Se não for, podemos habilitá-la na cena.

Resultado:



---

4. Tampo da mesa discretizado (Autores: Henrique Soares & Marcos Vinicius Araujo)

Para criação do tampo da mesa discretizado, criamos uma nova classe tampa_array, que possui todos os pontos dos triângulos e suas normais. Seguindo o mesmo raciocínio da criação da pirâmide e das outras formas que foram explicadas em aula, temos:

- tampa_array.h

```
inc1 > tampa_array.h > TampaArray
#include <memory>
class TampaArray;
typedef std::shared_ptr<TampaArray> TampaArrayPtr;

#ifndef TAMPA_ARRAY_H
#define TAMPA_ARRAY_H

#include "shape.h"

class TampaArray : public Shape {
  float m_coords[(5*4 + (7*7))*3]; // outras 5 faces, 4 pontos, 3 coordenadas
                                   //+ n de pontos da tampa ao quadrado * 3 coordenadas
  float m_normals[(5*4 + (7*7))*3];              // outras 5 faces, 4 pontos, 3 coordenadas
                                                 //+ n de pontos da base da tampa ao quadrado * 3 coordenadas
  unsigned int m_index[(5 + (7-1)*(7-1))*6 ]; // 5 faces * 6 pontos +
                                              //(n de pontos da base da tampa - 1) ao quadrado * 6
protected:
  TampaArray ();
public:
  static TampaArrayPtr Make ();
  virtual ~TampaArray ();
  virtual void Draw ();
};
#endif
```

Para facilitar a criação dos pontos, criamos um código em python que calcula automaticamente todas as posições corretas e a ordem dos índices do triângulos:

- cria_vertices.py

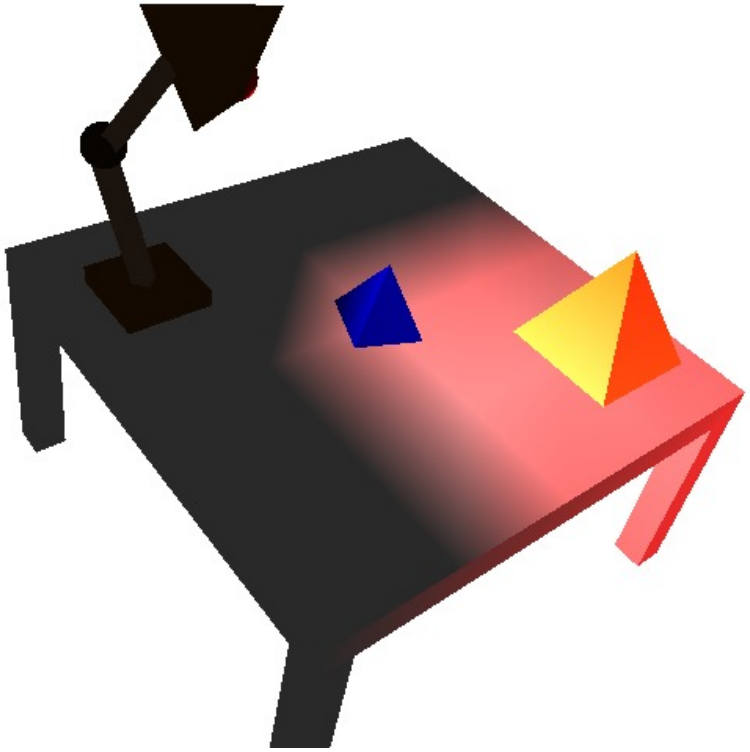
```
a = -0.5
b = 0.5
c = -0.5
d = 0.5
n = 3
for x in range(n):
    for y in range(n):
        step_x = 1/(n-1)*x
        step_y = 1/(n-1)*y
        #print(str(a+step_x)+'f,',str(1.0)+'f,',str(c+step_y)+'f,')
        print(str(a) + "+" + str(1.0)+"f/"+str(n-1)+'.0f*'+str(x)+'.0f,',str(1.0)+'f,', str(a) + "+" + str(1.0)+"f/"+str(n-1)+'.0f*'+str(y)+'.0f,')
i = 0
s = 20
for x in range(n):
    for y in range(n):
        if(i%n == n-1):
            i += 1
            continue
        if(i >= n*n-n):
            break
        print("{},{},{},{},{},{},".format(s+i,s+i+1,s+i+1+n,s+i,s+i+1+n,s+i+1+n-1))
        i += 1
```

A saída desse código é:

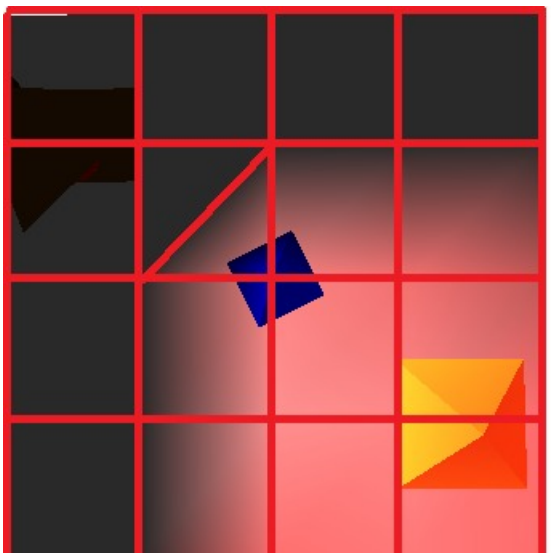
```
30,31,35,30,35,34,
>>>
=============== RESTART: D:\OneDrive\Área de Trabalho\aaaa.py ==========
-0.5+1.0f/2.0f*0.0f, 1.0f, -0.5+1.0f/2.0f*0.0f,
-0.5+1.0f/2.0f*0.0f, 1.0f, -0.5+1.0f/2.0f*1.0f,
-0.5+1.0f/2.0f*0.0f, 1.0f, -0.5+1.0f/2.0f*2.0f,
-0.5+1.0f/2.0f*1.0f, 1.0f, -0.5+1.0f/2.0f*0.0f,
-0.5+1.0f/2.0f*1.0f, 1.0f, -0.5+1.0f/2.0f*1.0f,
-0.5+1.0f/2.0f*1.0f, 1.0f, -0.5+1.0f/2.0f*2.0f,
-0.5+1.0f/2.0f*2.0f, 1.0f, -0.5+1.0f/2.0f*0.0f,
-0.5+1.0f/2.0f*2.0f, 1.0f, -0.5+1.0f/2.0f*1.0f,
-0.5+1.0f/2.0f*2.0f, 1.0f, -0.5+1.0f/2.0f*2.0f,
20,21,24,20,24,23,
21,22,25,21,25,24,
23,24,27,23,27,26,
24,25,28,24,28,27,
>>>
```

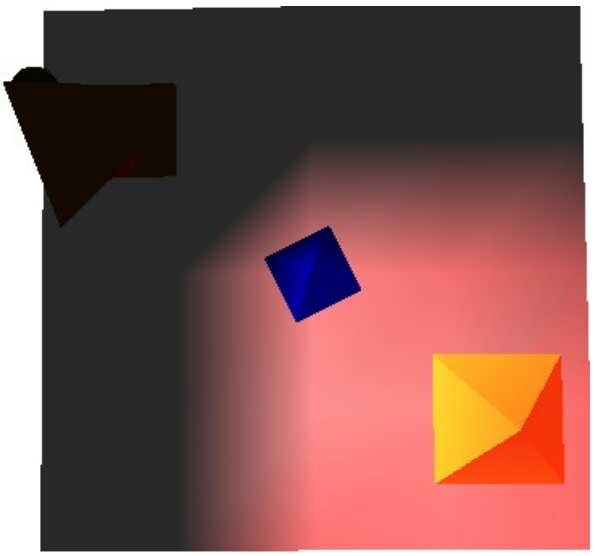
O código de criação dos vértices se encontra no arquivo zip enviado.

Após a criação dos vértices e sua inserção no código, podemos já testar se essa mesa discretizada funciona, utilizando a luz de spot criada acima. Segue o resultado:



Nessa imagem temos apenas a luz de spot ligada, para dar um maior destaque. E percebemos que apenas uma parte da mesa foi iluminada.





Esse teste foi realizado para 32 triângulos, porém mostraremos um teste mais robusto ainda no vídeo, com 98 triângulos!

5. **Engines para criação de cenas animadas (Autores: Marcos Vinicius Araujo & Henrique Soares)**

Para a criação da animação, foi criada uma classe engine_pyr muito parecida com egine, que herda da mesma:

- engine_pyr.h

```h
#include <memory>
class EnginePyr;
typedef std::shared_ptr<EnginePyr> EnginePyrPtr;

#ifndef ENGINE_PYR_H
#define ENGINE_PYR_H

#include "engine.h"
#include "transform.h"

class EnginePyr : public Engine
{
  float m_factor;
  TransformPtr m_pR;

protected:
  EnginePyr (TransformPtr pR);
public:
  static EnginePyrPtr Make (TransformPtr pR);
  ~EnginePyr ();
  void SetSpeedFactor (float factor);
  float GetSpeedFactor () const;
  void Update (float dt);
};

#endif
```

Em engine_pyr.cpp, alteramos basicamente o método Update, que rotaciona a pirâmide:

```cpp
void EnginePyr::Update (float dt)
{
  float ndays = dt / 5.0f * m_factor;
  m_pR->Rotate(360.0f*ndays,0.0f,1.0f,0.0f);
}
```

Após isso, a cada ciclo, a pirâmide será rotacionada por um fator, em seu eixo y.

6. Dois pontos de vista, dividindo a janela de desenho em duas regiões lado a lado (Autor: Henrique Soares)

Para criação das duas cenas, criamos mais 1 câmera e 1 arcaball e chamamos o método Render de Scene para renderizar as duas câmeras, como mostra o código abaixo:

```
static void display (GLFWwindow * win)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // clear window
  glViewport(0,0,600,600);
  scene->Render(camera);
  glViewport(600,0,600,600);
  scene->Render(camera2);
}
```

Para ambas as telas, colocamos um tamanho de 600x600 e exibimos uma do lado da outra.

- Criação da arcball:

```
arcball = camera->CreateArcball();
arcball2 = camera2->CreateArcball();
spot->SetAmbient(1.0f,0,0);

scene = Scene::Make(root,spot,light);
engine = EnginePyr::Make(trfPiramideR);
scene->AddEngine(engine);
```