PL12 - Color Picking and Bitmap Text

Notas para a componente prática de Computação Gráfica
Universidade do Minho
Carlos Brito and António Ramires

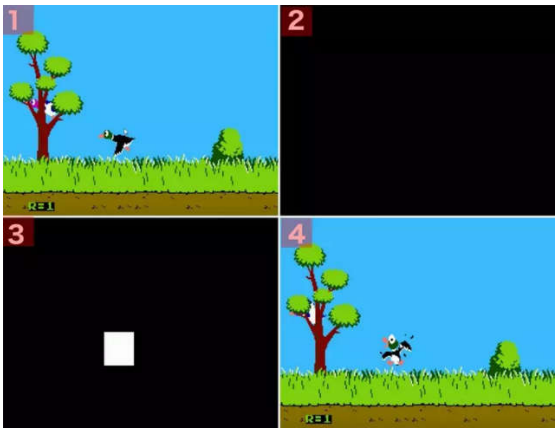
1. Introdução

Com este guião pretende-se adicionar uma interação simples entre o utilizador e a cena virtual: a seleção de um objeto 3D. Este problema é geralmente referido como picking e pode ser resolvido com várias abordagens. Duas das mais utilizadas são raycasting e color picking.

Raycasting é uma abordagem puramente matemática na qual é disparado um raio na de acordo com a posição/orientação da câmara atual e com as coordenadas do pixel que foi clicado. Depois de disparado, são feitos testes de intersecção do raio com os objetos em cena, que geralmente são aproximados por primitivas simples, como caixas, esferas e cilindros, normalmente designadas por bounding volumes. Este é um processo potencialmente custoso caso existam muitos objetos em cena, sendo normalmente utilizadas técnicas de particionamento de espaço, como Octrees, para evitar testes de interseção desnecessários.

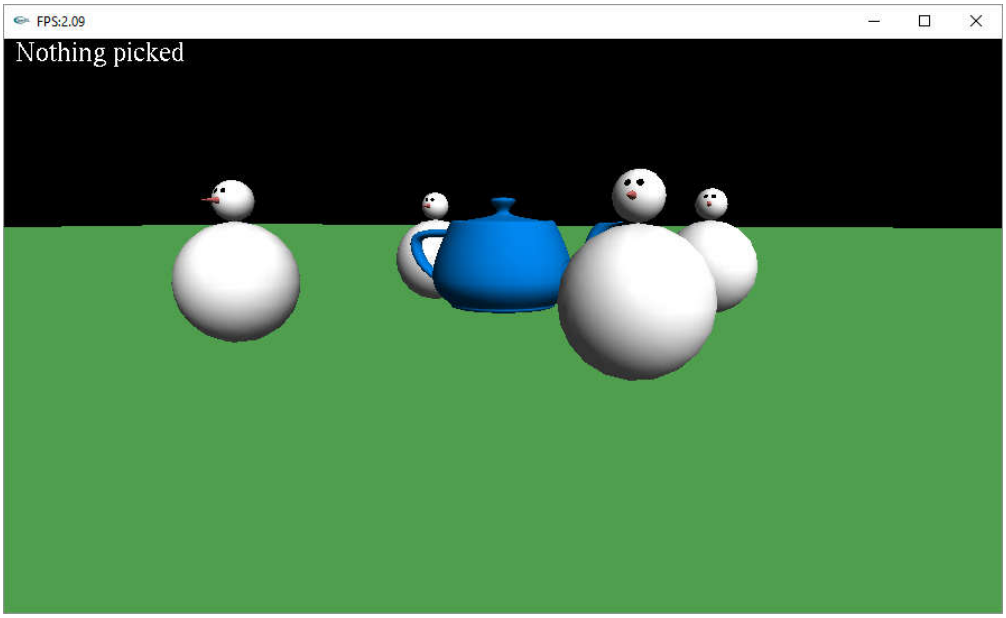
Nesta aula não iremos recorrer a raycasting, mas sim a color picking, que em termos gerais faz uso da API de desenho (neste caso OpenGL) para determinar qual o objeto que se encontra "por baixo" do cursor à base de cor. Na altura do clique a cena é redesenhada de forma a atribuir cores diferentes a cada objeto, recuperando esta cor a partir do pixel clicado sabemos exactamente que objeto foi selecionado.

O jogo clássico DuckHunt (1984) para a NES pode ser visto como um exemplo simples do uso de color picking. A pistola (zapper gun) funciona à base de um foto-díodo, que consegue determinar se atingiu uma região branca ou preta na televisão. Quando premido o gatilho toda a cena é redesenhada por breves instantes, sendo os patos desenhados como retângulos brancos. Caso o sensor detete esses mesmos retângulos então o tiro foi bem-sucedido.

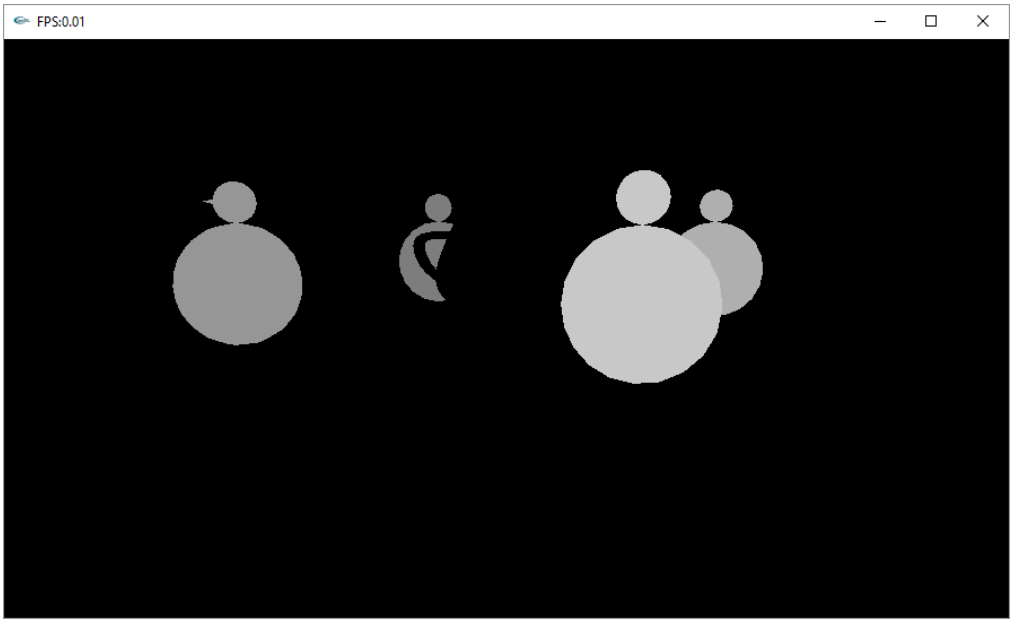


2. Abordagem

A cena fornecida consiste em 4 bonecos de neve:



De forma a descobrir qual deles foi clicado com o rato invocamos o OpenGL na altura do clique para redesenhar a cena, mas de forma codificada. Se desenharmos esta representação intermédia teria este aspeto:



Como é possível constatar na imagem codificada, cada boneco de neve possui apenas uma cor, ou seja, vai ser necessário desativar a iluminação (que produziria um gradiente) e também as texturas.

```
unsigned char picking(int x, int y)
{
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);

    ...
```

De seguida é necessário limpar o color buffer para podermos desenhar os bonecos de neve codificados. Iremos limpar apenas o color buffer, e não o depth buffer, visto que podemos usufruir dos valores de profundidade já presentes no depth buffer, que são corretos dado estarmos a desenhar a mesma cena.

```
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
gluLookAt(...)
```

Dado que não apagamos o depth buffer, se desenhássemos a cena tal e qual (com a função de comparação default GL_LESS para o depth test) então os bonecos de neve codificados seriam rejeitados por se encontrarem exactamente nas mesmas posições dos da frame atual, como tal temos de alterar a função de teste para GL_LEQUAL.

```
glDepthFunc(GL_LEQUAL);
```

Depois procedemos ao desenho da cena, tal como na função render scene, mas com os valores de cor codificados por boneco-de-neve.

Por último temos de recuperar a cor do pixel que foi clicado, podemos fazer isso recorrendo à função glReadPixels, juntamente com as coordenadas do rato, obtidas pelo GLUT. No entanto em OpenGL o eixo dos y encontra-se invertido.

```
unsigned char res[4];
GLint viewport[4];

glGetIntegerv(GL_VIEWPORT, viewport);
glReadPixels(x, viewport[3] - y, 1, 1, GL_RGBA, GL_UNSIGNED_BYTE, res);
```

Concluímos a abordagem reativando a texturas e luz, e devolvendo a componente R do pixel clicado:

```
glDepthFunc(GL_LESS);
glEnable(GL_LIGHTING);
glEnable(GL_TEXTURE_2D);

return res[0];
```

3. Desenho de Texto com bitmap fonts em glut

Para o desenho de texto, a biblioteca GLUT já disponibiliza um conjunto predefinido de tipos de letra (fonts), tal como uma API que desenha o texto já tomando em conta o espaçamento variável entre caracteres.

Até agora temos vindo a utilizar projeção perspetiva, na qual o volume de visualização é uma pirâmide e como consequência objetos mais distantes são reduzidos proporcionalmente. No entanto, quando estamos a desenhar texto 2D na janela (como uma interface de utilizador) não estamos interessados em utilizar projeção perspetiva, mas sim projeção ortogonal, cujo volume de visualização é uma caixa e a componente z é essencialmente ignorada. Neste tipo de projeção podemos definir diretamente a largura e altura da caixa, e regra geral escolhemo-los de forma a fazer coincidir as unidades de desenho OpenGL com as dimensões da janela em pixéis. Desta forma podemos posicionar o texto relativamente ao canto da janela, utilizando pixéis como unidades.

```
void renderText(const std::string& text) {

// Guardar a projeção anterior
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();

// Projecção ortogonal para que as coordenadas de desenho coincidam com o tamanho da
janela em pixeis

gluOrtho2D(0, w, h, 0);
glMatrixMode(GL_MODELVIEW);

glPushMatrix();
glLoadIdentity();

void* font = GLUT_BITMAP_HELVETICA_18;

// Centrar o texto, calculando a dimensão da mensagem em pixeis

float textw = glutBitmapLength(font, (unsigned char*) text.c_str());
glRasterPos2d(w/2 - textw/2, 24); // text position in pixels

// Ignorar profundidade
glDisable(GL_DEPTH_TEST);

// Desenhar a mensagem, caracter a caracter
for (char c : text)
{
    glutBitmapCharacter(font, c);
}

// Restaurar as matrizes anteriores
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
glPopMatrix();

glEnable(GL_DEPTH_TEST);

}
```