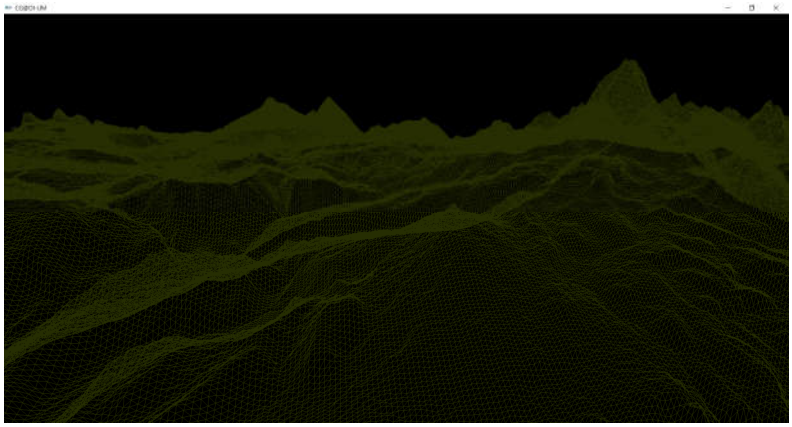
PL10 - Terrain

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

1. Introdução

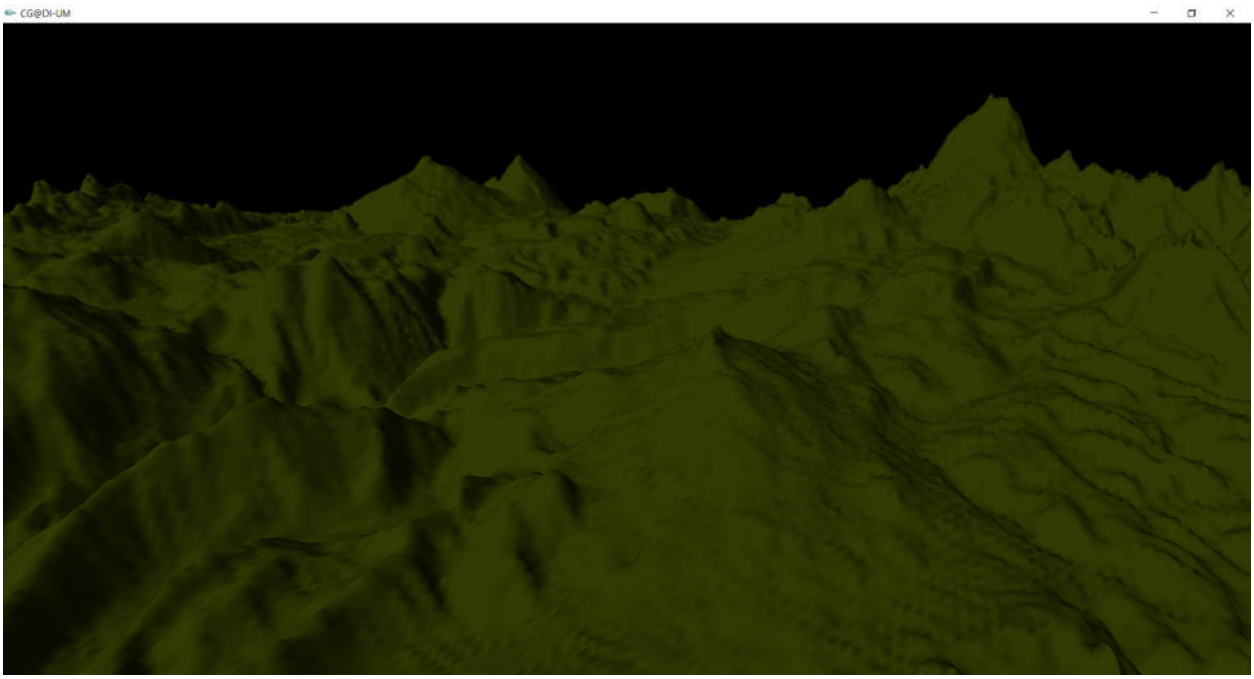
Com este guião pretende-se explorar a adição de normais e coordenadas de textura na construção de um terreno. Em qualquer aplicação de computação gráfica, o uso de normais é essencial para o cálculo da iluminação. Podemos constatar a importância deste efeito na seguinte sequência de renders:

Na imagen, sem normais e desenhadas com GL_FILL, nada é distinguível, a não ser que desenhemos com linhas:

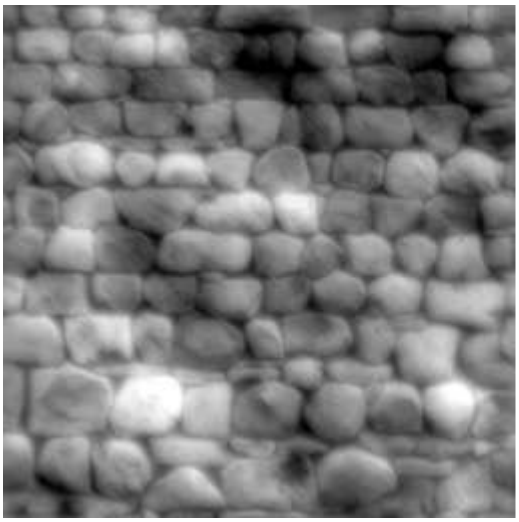


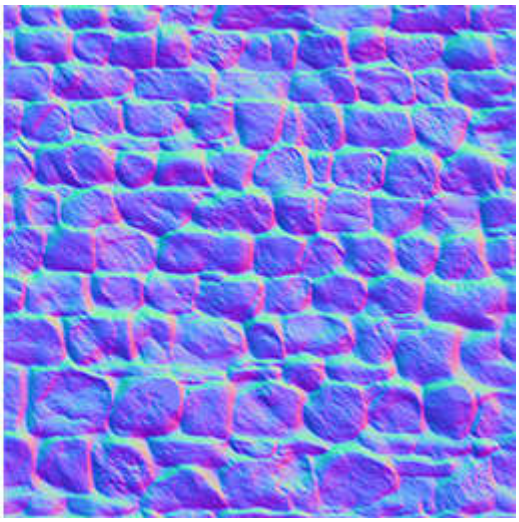
Através da “densidade” de linhas conseguimos ter alguma perceção da forma do terreno, no entanto, falta a iluminação:

Com iluminação apercebemo-nos dos vários detalhes que constituem o terreno.



Por vezes as normais são fornecidas numa imagem separada ao heightmap, conhecidas como normal maps:





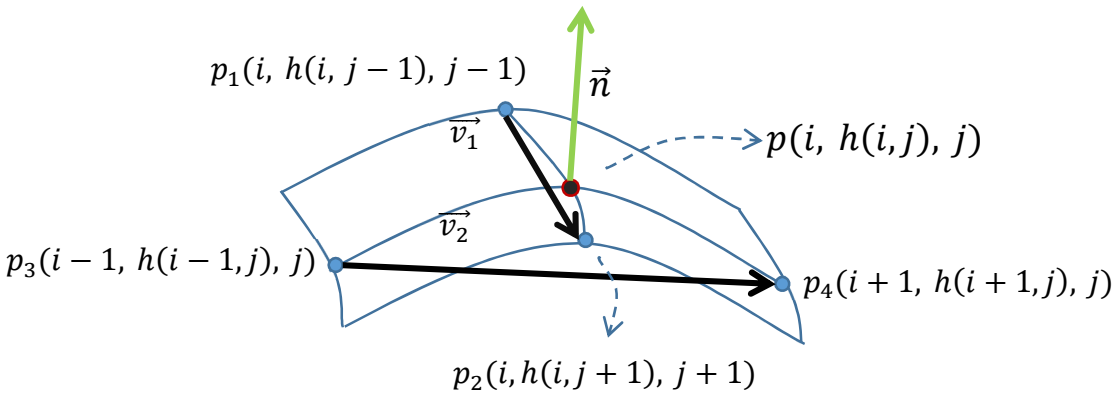
A diferença neste caso é que na imagem greyscale à esquerda os valores de intensidade de cada pixel são interpretados como uma altura, tal como na aula P06, e na imagem (a cores) à direita os valores RGB de cada pixel são interpretados como vetores direção da normal (R=x, G=y, B=z). De notar que a imagem é

maioritariamente azul, pois nesta imagem as direções das normais apontam maioritariamente na direção positiva do eixo dos z, ou seja, o canal B tenderá a ter valores mais altos.

Nesta ficha, não iremos recorrer a *normal maps*, mas sim calcular os vetores normais a partir do mapa de alturas. Isto pode ser feito com recurso a aproximações das derivadas parciais da imagem.

2. Cálculo das Normais

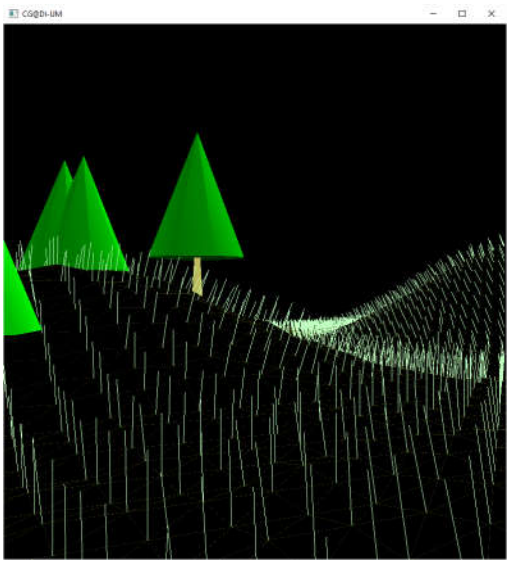
Consideremos uma pequena zona do terreno. As normais podem ser obtidas através das derivadas parciais, tal como no caso das superfícies de Bezier. A secante a uma superfície é uma aproximação à normal, sendo que no caso dos terrenos esta aproximação funciona bastante bem. O cálculo das secantes pode ser realizado recorrendo aos pontos vizinhos do ponto em questão, ou seja os pontos p_1, p_2, p_3 e p_4 da figura seguinte.



De notar que os vetores normais devem estar sempre **normalizados**, ou seja estes vectores devem ter comprimento 1. Sendo assim, a formulação desta operação torna-se:

$$\overrightarrow{v_1} = p_2 - p_1 \qquad \overrightarrow{v_2} = p_4 - p_3 \qquad \vec{n} = \frac{\overrightarrow{v_1} \times \overrightarrow{v_2}}{|\overrightarrow{v_1} \times \overrightarrow{v_2}|}$$

Na figura seguinte vemos as normais desenhadas como linhas.

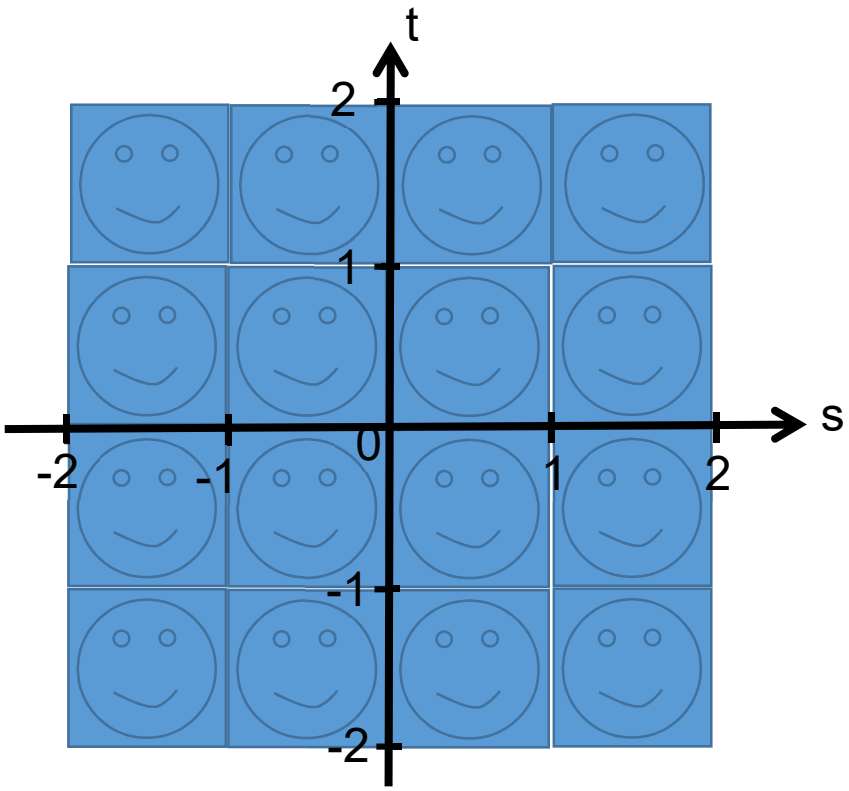


Tal como na aula 9, para desenhar o terreno com normais temos de criar um buffer adicional para as armazenar, tendo o cuidado de atribuir a normal correta a cada vértice, e ativar o respetivo client state. Consultar o guião da aula 9 para os detalhes da implementação da iluminação.

3. Texturas

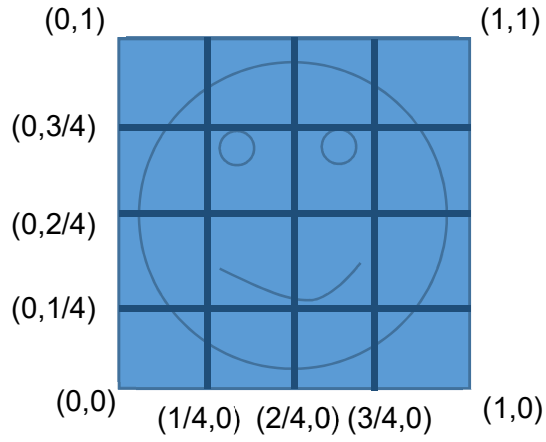
Tal como no caso das normais, é necessário atribuir a cada vértice uma coordenada de textura.

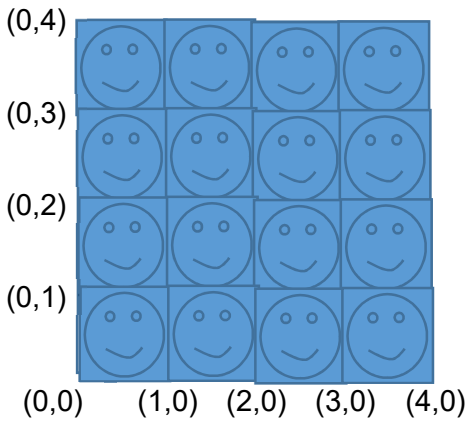
Por omissão a textura repete-se no espaço textura, tal como ilustrado na figura seguinte:



Vamos considerar dois modelos simples no caso do terreno. Estes modelos podem ser referidos como *Tiled* e *Streched.*

No modelo *Streched*, o objetivo é cobrir todo o terreno com apenas uma textura, enquanto no modelo *Tiled* a textura é repetida.





Para o guião pretende-se usar o modelo *Tiled*, sendo a textura repetida por cada célula. Assumindo que as células da grelha têm comprimento e largura iguais a 1, as coordenadas de textura pode assumir os valores das componentes x e z da grelha.

A utilização de texturas implica um procedimento semelhante às normais, ou seja, é necessário activar esta funcionalidade, e o respectivo *client state*.

É também necessário criar um novo buffer para armazenar as coordenadas de textura.

4. Implementação em OpenGL

Na fase de inicialização, é necessário activar as funcionalidades e os respectivos *client states*.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_TEXTURE_2D);
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
```

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

A preparação do terreno envolve agora três buffers: posições, normais e coordenadas de textura.

```
void prepareTerrain() {
      std::vector<float> position, normal, texCoord;
      for (int i = 1; i < imageWidth - 2 ; i++) {
          for(int j = 1 ; j < imageWidth -1; j++) {
               // fill arrays for position,
               // normal and texcoord to create strips...
          }
      }
      glGenBuffers(3, buffers);
      glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
      glBufferData(GL_ARRAY_BUFFER, position.size() * sizeof(float),
      &(position[0]),GL_STATIC_DRAW);
      glBindBuffer(GL_ARRAY_BUFFER, buffers[1]);
      glBufferData(GL_ARRAY_BUFFER, normal.size() * sizeof(float),
      &(normal[0]),GL_STATIC_DRAW);
      glBindBuffer(GL_ARRAY_BUFFER, buffers[2]);
      glBufferData(GL_ARRAY_BUFFER, texCoord.size() * sizeof(float),
                                    &(texCoord[0]),GL_STATIC_DRAW);
}
```

Todos os passo apresentados acima fazem parte da inicialização e devem ser executados uma única vez.

A função que desenha o terreno é semelhante à utilizada para desenhar o terreno da aula 9, sendo necessário adicionar o código correspondente à semântica dos buffers

```
void renderTerrain() {
      GLfloat white[] = {1.0f, 1.0f, 0.0f, 1.0f};
      glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
```

```
        glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
        glVertexPointer(3, GL_FLOAT, 0, 0);

        glBindBuffer(GL_ARRAY_BUFFER, buffers[1]);
        glNormalPointer(GL_FLOAT, 0, 0);

        glBindBuffer(GL_ARRAY_BUFFER, buffers[2]);
        glTexCoordPointer(2, GL_FLOAT, 0, 0);

        for (int i = 1; i < imageWidth - 2; i++) {
              glDrawArrays(GL_TRIANGLE_STRIP,
                  (imageWidth-2) * 2 * i, (imageWidth-2) * 2);
        }
}
```