Universidade Lusófona de Humanidades e Tecnologias

ULHT260-2267
Licenciatura em Engenharia Informática
Computação Gráfica
Nuno Cruz Garcia – p6040@ulusofona.pt

Aula Prática 05

Temas

Nas últimas aulas temos trabalhado com superfícies planas, triângulos e outros polígonos, ainda que num espaço 3D. Esta ficha prática explora o *pipeline* gráfico do OpenGL usando objectos 3D. O objectivo é perceber os vários sistemas de coordenadas do *pipeline* gráfico.

O *input* do *vertex shader* são as coordenadas do objecto/polígono/etc., e o *output* são as coordenas dos vértices no intervalo [-1,1]. Em termos práticos, o tema desta ficha são estas transformações geométricas que acontecem no *vertex shader*.

O triângulo que normalmente serve de exemplo às fichas práticas tem as suas coordenadas definidas no intervalo [-1,1]. Estas coordenadas são chamadas de Coordenadas de Dispositivo Normalizadas (CDN), que são depois transformadas em coordenadas 2D do ecrã pelo rasterizador. O output do *vertex shader* espera coordenadas na forma CDN. Além disso, os vértices que estiverem dentro do intervalo [-1,1] são renderizados e os que estiverem fora são descartados.

Nem sempre é conveniente definir os objectos que compõem a cena 3D directamente em CDN, principalmente se houver muitos objectos e de grande complexidade. O habitual é definir o objecto de forma isolada, com as suas coordenadas locais (p.ex. ideia da última aula teórica, em que os componentes do objecto "casa" têm as suas coordenadas locais – quadrado -> janelas -> casa). Outro exemplo é a importação de modelos feitos em Blender ou outro software de modelação 3D, e que podem ter coordenadas muito variadas – a ser explorado numa futura aula prática.

A transformação de coordenadas locais em CDN passa por vários estágios, ou seja, são transformadas em vários sistemas de coordenadas intermédios que facilitam certo tipo de transformações.
Os 5 sistemas de coordenadas que vão ser discutidos são:

- Coordenadas do Objecto (Local space ou Object space)
- Coordenadas do Mundo (World space)
- Coordenadas da Câmara (View space ou Eye space)
- Coordenadas de Recorte (Clip space)
- Coordenadas de Dispositivo (Screen space)

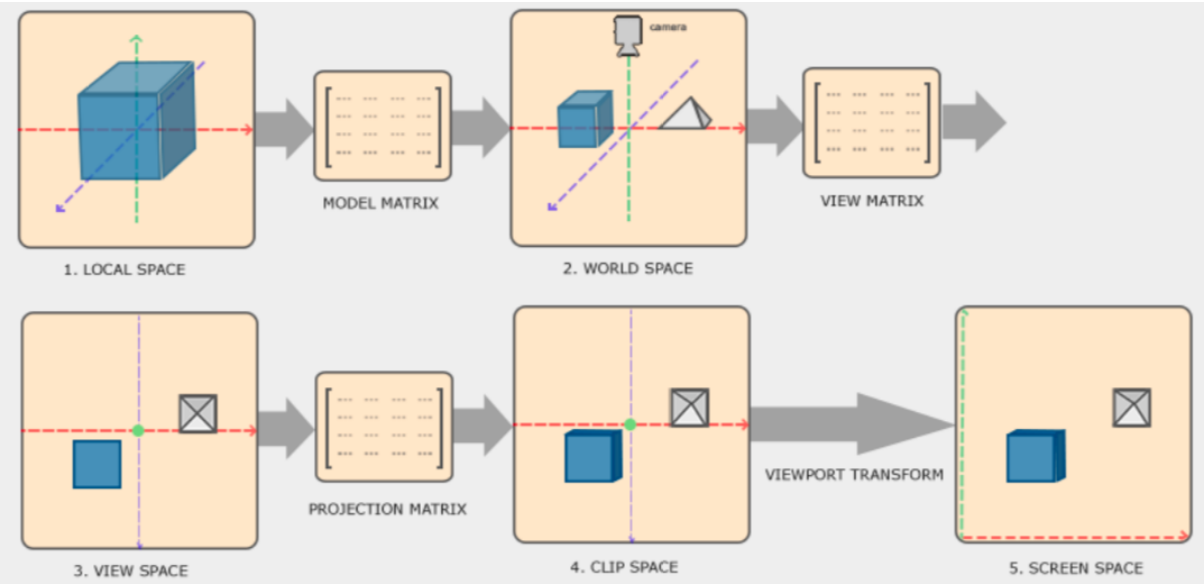


Figura 1 - https://learnopengl.com/Getting-started/Coordinate-Systems

A figura 1 descreve o seguinte processo:

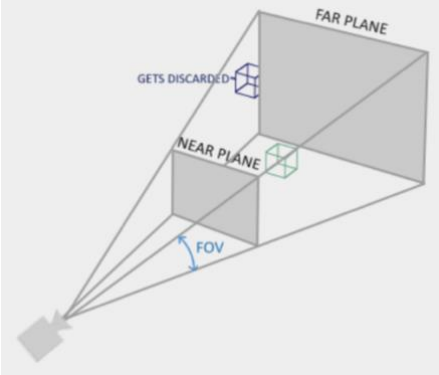
1. As Coordenadas do Objecto são as coordenadas locais do objecto relativamente à sua origem. São as primeiras coordenadas em que o objecto vive.
2. As Coordenadas do Mundo são coordenadas relativas a uma origem global de um espaço "mundo", onde vivem outros objectos da cena que se quer representar. Esta transformação é feita através da Matriz Modelo.
3. As Coordenadas do Mundo são transformadas em Coordenadas da Camâra, que são as coordenadas observadas pela câmara. Isto é feito pela Matriz de Visualização.
4. A transformação feita pela Matriz de Projecção transforma as coordenadas em Coordenadas de Recorte, que estão no intervalo [-1, 1]. Todos os vértices fora deste intervalo não serão visualizados.
5. A transformação de Viewport transforma as coordenadas de Recorte em coordenadas de Ecrã, que estão no intervalo definido em glViewport – o tamanho da janela do ecrã, em pixéis. Estas coordenadas são enviadas para o rasterizador, que as transforma em fragmentos (e que são coloridos posteriormente pelo *fragment shader*).

Algumas considerações sobre o que cada sistema de coordenadas oferece:
. Coordenadas Locais – Facilitam a modelação do objecto, sem ligação ao mundo exterior, ao local onde vão ser colocados na cena, ou a outros objectos. Cada objecto tem as suas coordenadas locais, e é definido em relação à sua própria origem.

Coordenadas Mundo – A matriz Modelo trata de posicionar um objecto no mundo. Pode ser a combinação de transformações geométricas como escalamento, rotação, translação, etc.

Coordenadas de Câmara (Eye space, camera space) – Série de rotações e translações, expressas pela matriz Modelo, que coloca os objectos em frente da câmara. É neste espaço que faz sentido pensar na deslocação da câmara pela cena 3D, sem alterar as posições globais dos objectos na cena, mas alterando apenas a posição da câmara.

Coordenadas de Recorte – Definimos a perspectiva com que se observa a cena. A analogia com uma câmara real é a seguinte: uma vez escolhida a posição da câmara do passo anterior, agora podemos definir a lente e o foco da câmara. Na prática, o resultado é um espaço da cena que será de facto considerado para renderizar, ou seja, que vértices devem ser visualizados e que vértices devem ser rejeitados. Regra geral esta matriz, chamada matriz de Projecção, não se altera.



Ou seja, para obter Coordenadas de Recorte a partir das coordenadas dos vértices:

$$V_{Recorte} = M_{projecção} \cdot M_{câmara} \cdot M_{modelo} \cdot V_{local}$$

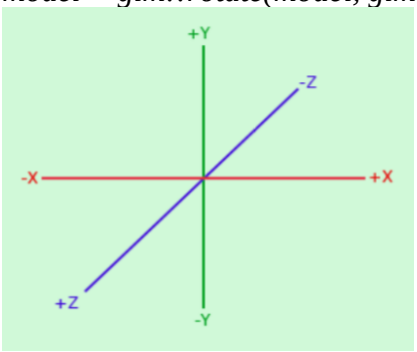
Onde V se refere às coordenas e M às matrizes. Repare-se na ordem inversa da multiplicação das matrizes: o pipeline começa com as coordenadas locais V_{local} às quais se aplica a matriz Modelo, de seguida a matriz de Visualização e de seguida a matriz de Projecção.

As coordenas $V_{Recorte}$ são assignadas à variável *gl_Position*, que é o output do Vertex Shader.

Exercícios

Considere o código que acompanha esta ficha.

Perceber a transformação do plano. Experimentar com diferentes valores
model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));



Se ao rodar em torno do eixo dos xx o plano se posiciona “sobre o chão”, que transformação o coloca na “parede da esquerda”?

Construir um cubo, definido por 36 vértices. Porquê 36 vértices? (6 faces * 1 quadrado) = (6 faces * 2 triângulos) = (6 * 2 * 3).

O que acontece de estranho?

Fazer enable ao Z_Buffer.
glEnable(GL_DEPTH_TEST);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

O que é preciso fazer para desenhar 10 cubos?
Que matriz deve ser usada para dispor os cubos em posições diferentes da cena 3D?

Experimentar com diferentes locais da câmara:
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
Que valores deve ter se quiser deslocar a câmara ligeiramente para cima e para a esquerda?

e diferentes perspectivas:
projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
O que acontece se o campo de visão aumentar?