

JSR-184, API Gráfica 3D para J2ME

André Ribeiro da Silva Cravo
Leiria, Portugal
ei08465@student.estg.ipleiria.pt

Marco Paulo Martins Pedro
Leiria, Portugal
ei08110@student.estg.ipleiria.pt

Resumo

Este documento tem o intuito de passar em revista a API de gráficos 3D para J2ME, JSR-184. Será focada a API e uma spike solution desenvolvida de modo a mostrar algumas das potencialidades da API.

Palavras-chave

JAVA, J2ME, JSR-184, Mobile 3D Graphics API

INTRODUÇÃO

A JSR-184 é uma especificação Java que define uma API de gráficos tridimensionais para dispositivos móveis. A API encontram-se na forma de um pacote opcional que poderá ser utilizado com a versão 1.1 do *Mobile Information Device Profile* (MIDP) da *Connected Limited Device Configuration* (CLDC) [3]. Esta especificação surgiu devido à necessidade do uso de gráficos 3D em dispositivos móveis, algo que a arquitectura J2ME não disponibilizava. Apenas era possível desenhar primitivas gráficas 2D disponibilizadas pela classe *Graphics*; O OpenGL ES só que este era demasiado de baixo nível o que fazia com que para realizar tarefas relativamente simples fosse necessário produzir inúmeras linhas de código; O Java 3D API era demasiado grande e não era compatível com o MIDP.

No entanto não se deverá confundir a API Java 3D com a JSR-184 (Java Mobile 3D). A API Java 3D encontra-se destinada para ambientes de workstations enquanto que a JSR-184 é para dispositivos móveis com baixos recursos de hardware.

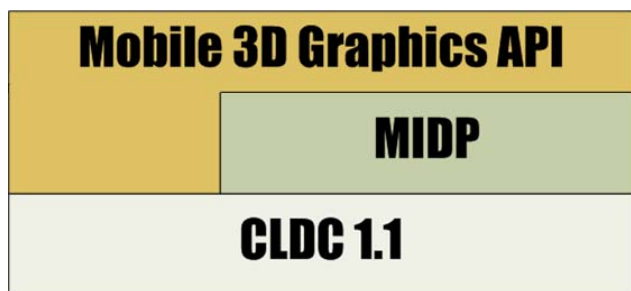


Figura 1. Pilha da API Mobile 3D

API JSR-184

A API disponibiliza um conjunto de classes e um motor para efectuar o *rendering* de modo a que o programador

possa construir ambientes e animações 3D capazes de serem executadas em tempo real. A JSR-184 foi desenhada tendo em consideração dispositivos móveis com poucos recursos de hardware, memória, processamento, ecrã e suporte para vírgula flutuante. Contudo, esta é escalável para dispositivos que disponibilizem um melhor e mais forte suporte para os itens acima referidos. A JSR-184 está preparada para poupar ao máximo os recursos de um dispositivo, reduzindo a RAM que é usada e por consequência a *garbage collection*. A sua implementação ocupa apenas 150KB.

A API disponibiliza dois modos de fazer uso dos gráficos 3D, sendo um de alto nível (*Retained-mode*) e outro de baixo nível (*Immediate-mode*).

O uso desta API estende-se a áreas como Jogos, Visualização de mapas, desenvolvimento de interfaces gráficas, mensagens animadas, *screen savers*, etc.

Requisitos da API-184

A especificação diz que a API deve respeitar os seguintes requisitos:

- Suporte para alto nível (*Retained-mode*) e baixo nível (*Immediate-mode*), e combinação entre os dois modos.
- Não deve conter partes opcionais. Deve ser implementada na totalidade.
- De modo a reduzir a quantidade de programação necessária para a sua utilização, a API deve incluir tipos especiais como *Meshes*, *Textures*, etc.
- Dados deverão ser codificados em binário de modo a ocupar menos espaço e ser mais fácil de enviar.
- Deverá ser possível implementar a API de modo eficiente sobre o OpenGL ES, sem ser necessário recorrer a hardware para processar operações de vírgula flutuante.
- A API deve usar o tipo *float* da linguagem Java (para coordenadas, transformações e etc.) e não introduzir nenhum outro tipo de dados na sua vez. Por consequência, qualquer dispositivo móvel que use esta API deverá basear-se na versão 1.1 do CLDC (versão onde o tipo *float* já existe).

- Visto que o uso da aritmética de inteiros é difícil e não está livre de erros, deverão ser usados valores de virgula flutuante sempre que possível.
- A API deverá ser implementada em 150KB de um dispositivo móvel.
- A API deve fornecer *garbage collection*.
- A API deve inter operar correctamente com outras API do Java especialmente com a MIDP.

Classes da API JSR 184

A API disponibiliza um total de 30 classes. Este conjunto de classes disponibiliza bastantes operações 3D (tendo em conta o tamanho da API e dispositivos onde esta irá correr). Permite criar ambientes com o efeito de nevoeiro, aplicar texturas, usar luzes, atribuir as qualidades de materiais (ex. Definir o tipo de reacção que o material irá ter a nível de propriedades difusa, especular e ambiente, quando este é incidido por uma luz) aos objectos, sombras, etc. [2]

Tabela 1. Classes da API JSR 184

| Classe | Descrição |
|---------------------|--|
| AnimationController | Gere a localização e velocidade de um conjunto de objectos associados a uma sequência de animação. |
| AnimationTrack | Contém informação que controla uma única propriedade de animação num objecto alvo. Uma sequência de animação consiste num conjunto de <i>AnimationTracks</i> controladas por um <i>AnimationController</i> . |
| Appearance | Conjunto de componentes que definem atributos usados no acto de <i>rendering</i> . Os atributos descrevem as características materiais, os polígonos, como é que devem ser misturados na cena e efeitos de nevoeiro que possam existir. Define também características relativas ao mapeamento de texturas e as imagens envolvidas. |
| Background | Usada para especificar a cor ou imagem de fundo que é usada para limpa ou preencher o fundo do <i>viewport</i> . |
| Camera | Nodo que define o vista sobre a cena. Usada para renderizar a cena de 3D para 2D. Define também quais o elementos que são visíveis na cena (<i>clipping</i>). |
| CompositingMode | É um componente da classe <i>Appearance</i> que contem os atributos necessários de modo a efectuar operações de composição sobre <i>pixels</i> . |
| Fog | É um componente da classe <i>Appearance</i> que contem os atributos necessários para aplicar o efeito de nevoeiro aos |

| | |
|------------------|---|
| | <i>pixels</i> . |
| Graphics3D | <i>Singleton Graphics 3D context</i> que serve para efectuar o <i>rendering</i> de uma imagem. Este objecto é ligado a um alvo no qual é efectuado o <i>rendering</i> , que poderá ser um <i>Canvas</i> , <i>mutable Image</i> ou um <i>CustomItem</i> . Esta classe efectua todo o processo de desenho da API JSR 184. |
| Group | Nodo que guarda de uma forma não ordenada outros nodos como filhos. |
| Image2D | Representa uma imagem 2D que pode ser usada como textura, imagem de fundo ou <i>sprite</i> . Existem dois tipos de imagens: <i>mutable images</i> , que podem ser actualizadas em qualquer altura; <i>immutable images</i> , que são definidas durante a construção e não podem ser alteradas mais tarde. |
| IndexBuffer | Define a forma como os vértices são ligados para formar um objecto geométrico. |
| KeyFrameSequence | Encapsula uma animação na forma de uma sequência de time-stamped, vector-valued keyframes, na qual cada uma representa o estado da animação num dado instante. Pode ser associada a vários objectos da animação. |
| Light | Classe que representa diferentes tipos de fontes de luz. |
| Loader | Classe que permite efectuar o download e desserialização de nodos e cenas que se encontram em ficheiros M3G. O uso destes ficheiros é geralmente a maneira mais conveniente para uma aplicação carregar cenas 3D. |
| Material | Classe que encapsula atributos usados em cálculos para iluminação. É um componente da classe <i>Appearance</i> . |
| Mesh | Nodo que representa um objecto 3D definido por uma superfície poligonal que por sua vez é constituída por <i>triangle strips</i> . Representa um polígono rígido, enquanto que as subclasses <i>MorphingMesh</i> e <i>SkinnedMesh</i> têm capacidades para transformar os vértices do polígono que representam, independentemente uns dos outros. |
| MorphingMesh | Nodo que representa um <i>vertex-morphing mesh</i> de um polígono. |
| Node | Classe abstracta para todos os tipos de nodos. <i>Camera</i> , <i>Group</i> , <i>Light</i> , <i>Mesh</i> e <i>Sprite3D</i> são sub-classes do <i>Node</i> . |
| Object3D | Classe abstracta para todos os objectos que possam fazer parte de um mundo 3D. |
| PolygonMode | Encapsula atributo a nível do polígono, incluindo definições sobre <i>face culling</i> |

| | |
|--------------------|---|
| | (<i>back/front</i>), cálculos para iluminação, correcções de perspectiva, sombras e polygon winding. É um componente da classe <i>Appearance</i> . |
| RayIntersection | É um objecto que contém raios adicionados pelo método <i>pick</i> da classe <i>Group</i> . Guarda a referência para uma <i>Mesh</i> ou <i>Sprite3D</i> que intersectam cada raio e a informação relativa ao ponto de intersecção. |
| SkinnedMesh | Nodo que representa <i>Mesh</i> poligonal capaz de sofrer alterações nos seus vértices de forma independente uns dos outros. |
| Sprite3D | Representa uma imagem 2D com posição 3D. |
| Texture2D | Componente que encapsula uma textura 2D (imagem) e atributos que especificam como é que a imagem deve ser aplicada a uma <i>submesh</i> . É um componente da classe <i>Appearance</i> . |
| Transform | Representa uma matriz de 4x4 com <i>floats</i> que definem uma transformação. |
| Transformable | Classe abstracta das quais estendem as classes <i>Node</i> e <i>Texture2D</i> . Esta classe define os métodos para a manipulação de transformações em nodos e texturas. |
| TriangleStripArray | Define um <i>array</i> de <i>triangle strips</i> . Os primeiros três vértices definem o primeiro triângulo e o próximo vértice juntamente com os dois últimos formam o segundo triângulo e assim sucessivamente. Ex.: $ts = \{1, 2, 3, 4\} \rightarrow t1 = \{1, 2, 3\}$ e $t2 = \{2, 3, 4\}$ |
| VertexBuffer | Guarda referências para <i>VertexArrays</i> que contém informação acerca das coordenadas das posições, normais, cores ou texturas. |
| World | É um nodo que estende da classe <i>Group</i> e representa o <i>top-level container</i> de uma cena. Numa cena todos os outros nodos encontram-se ligados uns aos outros pelo nodo <i>World</i> (<i>root node</i>). |

Dever-se-á referir que a API JSR-184 ainda não disponibiliza mecanismos para detectar colisões entre *sprites* ou modelos 3D.

Quando se pretende misturar gráficos 2D com 3D, deve-se primeiro processar todo o tipo de gráficos 3D, e, depois de libertar o contexto 2D (Ex. *Graphics*) pode-se então processar os gráficos 2D. Exemplo:

```
protected void paint(Graphics g) {
    //Processar os gráficos 3D
    g3d.bindTarget(g);
    g3d.render(world);
    g3d.releaseTarget();
}
```

```
//Processar os gráficos 2D
g.drawArc(x, y, 35, 35, 0, 360);
...
}
```

Retained-Mode

Particularmente usado em jogos, este modo de alto nível carrega um ficheiro no formato JSR-184 (*m3g*), contendo o mundo 3D com todos os seus componentes, para a aplicação. O mundo é normalmente criado recorrendo a software de modelação 3D e depois exportada para o ficheiro *m3g*. A aplicação pode aceder aos diversos componentes do mundo carregada, animar objectos ou criar efeitos [1].

Exemplo do uso do modo:

```
public class MyCanvas extends Canvas
    Graphics3D g3d;
    World world;

    public MyCanvas() {
        g3d = Graphics3D.create();
        Object root[] = Loader.load("world.m3g");
        world = root[0];
    }

    protected void paint(Graphics g) {
        g3d.bindTarget(g);
        g3d.render(world);
        g3d.releaseTarget();
    }
}
```

Immediate-Mode

Apropriando para aplicações que geram gráficos 3D de forma algorítmica (Aplicações de visualização científica, gráficos, etc.), este modo permite ao programador manipular em baixo nível os gráficos 3D [1]. Neste modo o programador necessita de definir explicitamente pelo menos uma *câmara* e limpar os *buffers* de cor e profundidade.

A primitiva gráfica usada neste modo é o triângulo e estes são expostos ao *rendering* na forma de *triangle strips*. Na API não existem primitivas que desenhem formas geométricas predefinidas como no caso do OpenGL Utility Toolkit (GLUT) que disponibilizava métodos para desenhar esferas, torus, cubos e etc.

Neste modo, uma das classes mais importantes é a *VertexBuffer*. Como já foi referido anteriormente, nesta classe serve para guardar informação acerca de um objecto 3D. Ex.: para definir um cubo poder-se-ia fazer o seguinte:

```
short[] coord = { 5, 5, 5, -5, 5, 5, -5, 5, -5, -5, 5, //frente
```

```

...}
VertexArray vertArray = new VertexArray(coord.length/3,
3, 2);
vertArray.set(0, coord.length/3, coord);
int[] stripLen = { 4, 4, 4, 4, 4, 4 };
VertexBuffer cubo = new VertexBuffer();
cubo.setPositions(vertArray, 1.0f, null);
cuboIB = new TriangleStripArray( 0, stripLen );
...
g3d.render( cubo, cuboIB, new Appearance(), new Transform() );

```

Retained-mode VS. Immediate-mode

Eis alguns contrastes entre os dois modos da API:

- *Retained mode* torna menos complicado para o programador desenvolver ambientes 3D complexos.
- *Retained mode* carrega um mundo 3D de uma só vez.
- *Immediate mode* permite um maior controlo sobre o *rendering* de um ambiente 3D, o que poderá ser bastante importante em dispositivos com recurso escassos.

ALTERNATIVAS À API GRÁFICA 3D JSR 184

Para além desta API existem outras que permitem a criação de gráficos 3D em dispositivos móveis.

OpenGL ES é uma implementação independente da plataforma para sistemas embebidos (como por exemplo, dispositivos móveis) baseada no *OpenGL*. É uma API de baixo nível. [4]

Mascot Capsule Micro3D é uma extensão à API JSR 184. É uma implementação proprietário e da *HI Corporation*. É uma API bastante comum nos dispositivos da marca Sony Ericsson. É também uma implementação popular em países como o Japão e a Coreia. [5]

DXPAK é uma versão reduzida do DirectX para Windows CE. Esta API está apenas disponível para dispositivos que corram o Windows CE e para além de permitir o uso de gráficos 3D, também inclui capacidades para o *streaming* de áudio e vídeo. [6]

CUIDADOS E SUGESTÕES A TER NO DESENVOLVIMENTO DE APLICAÇÕES 3D

Ao desenvolver aplicações 3D é necessário que se tenham determinados cuidados de modo a obter uma melhor performance, principalmente em dispositivos cujo poder computacional é bastante reduzido quando comparados a *workstations*. Eis uma lista de alguns cuidados que se deverão ter:

- Usar vários tipos de detalhe na cena de modo a reduzir a carga no processador. Exemplo: um

objecto que se encontre distante num cenário poderá ser representado como um *sprite* e um objecto que esteja mais perto da como objectos 3D. Deste modo reduz-se a carga no processador.

- Reduzir a profundidade do mundo 3D de modo a minimizar os cálculos computacionais.
- No caso de se estar na presença de, por exemplo, um jogo do tipo *first person* e se pretender que o nosso jogador olhe para outros lados e mude de direcção, deve-se efectuar as transformações (ex.: rotações) sobre os objectos do cenário em vez de ser pela câmara. Isto porque efectuar as transformações sobre os objectos do cenário é menos pesado do que mudar o ângulo de visão da câmara.
- Não utilizar efeitos que sejam desnecessários. Existem efeitos como o nevoeiro que são computacionalmente dispendiosos.
- Utilizar o modo *retained* em alternativa ao modo *immediate* para efectuar o *rendering* de mundos 3D. Deste modo o mundo é carregado de uma só vez o que diminui a troca de informação entre a *virtual machine* e o sistema operativo do dispositivo.
- Compreender o sistema de vídeo do dispositivo. Saber quanta VRAM¹ o dispositivo disponibiliza pode ajudar a otimizar uma aplicação 3D.
- Aproveitar as características únicas de cada dispositivo. Num telemóvel pode-se, usar a luz de fundo para simular os relâmpagos de uma tempestade numa animação, usar a vibração para simular uma explosão ou tirar partido da rede para jogar com outros jogadores.
- Realização de *benchmarks*. Estes ajudam a compreender a performance dos dispositivos e por consequência o que é que eles poderão oferecer ao programador. Essa informação poderá permitir a optimização das aplicações.

SPIKE SOLUTION

Como *spike solution* foi desenvolvido uma aplicação que demonstra o funcionamento de ambos os modos. Para o modo *immediate*, a aplicação constrói um cubo e permite ao utilizador efectuar 3 tipos de transformações (rotação, translação e mudança de escala). Tal como foi referido anteriormente, foi necessária a definição de uma câmara e a limpeza dos buffers de profundidade e cor. Foi definida uma luz para iluminar o cubo. Para construir o cubo, foram definidos arrays com as posições dos vértices, cores e valores das normais para cálculos de iluminação. Foi definido o

¹ VRAM – Vídeo RAM. Memória utilizada para aplicações gráficas

material para o cubo, onde se especifica a reacção à luz que é emitida sobre o cubo. No modo *retained*, recorreu-se a aplicação de modelação *3D Studio Max*, onde foi gerado um cenário que depois foi exportado para o formato m3g. Depois de obter o ficheiro, este foi carregado recorrendo ao método *load()* da classe *Loader*. A partir desse método obtém-se o nodo raiz (*World*) que depois é renderizado recorrendo ao *render()* da classe *Graphics3D*.

O código do da *spike solution* encontra-se em anexo num ficheiro compactado.

DISPOSITIVOS QUE SUPORTAM A TECNOLOGIA

Sony-Ericsson

- K700
- S700
- K500
- Z500
- V800
- F500

Nokia

- 9500

Siemens

- S65
- S66
- S6C
- S6V
- SK65
- SL65
- SP65

CONCLUSÃO

Com o lançamento da API JSR-184 o mercado dos jogos para telemóveis ganhou um novo impulso, mas será que os telemóveis que já usufruem desta nova tecnologia vão substituir as consolas de jogos!? As consolas de jogos são dispositivos muito mais poderosos a nível de processamento e, a curto prazo, não serão substituídos por telemóveis. Isto porque um telemóvel não tem capacidades elevadas de processamento que uma consola disponibiliza. A API vem

resolver a falta de gráficos 3D nos dispositivos, mas isto só por si não resolve o problema do poder de processamento, mais ainda quando se fala de jogos. No entanto quando se pretende usar os gráficos 3D para a criação de *screen savers* ou aplicações que pretendem gerar conteúdo 3D numa forma diferente da dos jogos (visualização científica, geração de gráficos, etc.) esta API porta-se bastante bem.

São as limitações de hardware que acabam por impor limitações na API, mas, para os dispositivos que se encontram no mercado com suporte para a API, na sua generalidade, os resultados obtidos são relativamente bons. O desenvolvimento de aplicações com esta nova API torna-as muito mais agradáveis e interessante para o utilizador do que apenas com os gráficos 2D.

A nível do programador esta API é também uma aposta ganha visto que para gerar conteúdo 3D, são requeridas muito menos linhas de código do que programando com OpenGL ES, o que faz com que o tamanho da aplicação também seja reduzido.

AGRADECIMENTOS

Docente Catarina Reis

REFERENCES

- [1] Wireless Toolkit User's Guide – Chapter 9, Working With Mobile 3D Graphics
<<http://java.sun.com/j2me/docs/wtk2.2/docs/UserGuide-html/mobile3d.html>>
- [2] Mobile 3D Graphics and Java Application Development for Sony Ericsson Phones.
- [3] Getting Started With the Mobile 3D Graphics API for J2ME (JSR 184)
<<http://developers.sun.com/techtoc/mobility/apis/articles/3dgraphics/>>
- [4] OpenGL ES Overview
<<http://www.khronos.org/opengles/>>
- [5] HI Corporation Mascot Capsule
<<http://www.mascotcapsule.com/en/index.html>>
- [6] Microsoft Introduces DirectX for Windows CE
<<http://www.microsoft.com/presspass/press/2000/Feb00/DxpackPR.asp>>