Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

Bibliotecas 3D
para
Plataformas Móveis

Projecto realizado por:
990310 – José Porto

Índice

1 Introdução

Os dispositivos móveis estão cada vez mais presentes no nosso quotidiano. São cada vez menos as pessoas actualmente que não possuem telemóveis e cada vez mais as que possuem PDA's e outros dispositivos portáteis. Se é verdade que até há bem pouco tempo estes dispositivos tinham capacidades multimédia pouco evoluídas, também o é que são raros os dispositivos recentes que não possuam algumas capacidades multimédia, como ecrãs a cores com boas resoluções, boas capacidades de memória e cada vez mais capacidade de processamento. Estas capacidades são em alguns casos equivalentes aos primeiros computadores multimédia de mesa com capacidade de tratar gráficos 3D, com processadores com velocidades superiores a 100 MHz, capacidades gráficas razoáveis e alguma memória interna.

Como resultado desta evolução crescente, cada vez mais se torna viável a concepção de aplicações 3D nos dispositivos móveis, que podem apresentar desde mapas tridimensionais aos jogos. Como prova de que o desenvolvimento destas aplicações é levado a sério, surge o anúncio por parte de vários fabricantes de *hardware* gráfico (NVIDIA, ATI, PowerVR, etc.) de *chipsets* gráficos a serem desenvolvidos especialmente para plataformas móveis.

Existe assim a necessidade da existência bibliotecas gráficas 3D especializadas em plataformas móveis, tal como aconteceu nos PC's de secretária há uns anos atrás. Já existem várias bibliotecas especializadas em plataformas móveis, assim como vários derivados dessas bibliotecas, que são designados por *middleware*. O estado actual e o futuro destas bibliotecas, assim como o *middleware* nelas baseado, são o mote das páginas que se seguem.

2 Situação Actual

Os sistemas portáteis actualmente estão numa fase de transição. Estão a evoluir para além das suas capacidades originais e a transformarem-se em verdadeiras máquinas multimédia. No entanto, por muito que evoluam, as plataformas móveis nunca conseguirão estar a par dos sistemas *desktop*, devido às suas limitações energéticas e de tamanho. É praticamente impossível ter o poder de uma plataforma *desktop* numa plataforma móvel limitada por uma bateria e por um processador de baixo consumo. No entanto, as plataformas móveis actuais já estão ao nível dos primeiros PC's com capacidades multimédia, e a próxima geração promete estar ao nível das primeiras plataformas *desktop* com capacidade de aceleração gráfica 2D/3D. O quadro seguinte compara alguns sistemas portáteis actuais com várias plataformas PC.

	PDA's		***Smartphones*/Telemóveis**		**Computadores Pessoais**		
Produtos	HP iPAQ h5550 Pocket PC	Asus Mypal A730 Pocket PC	Sony Ericsson P910	Nokia N-Gage QD	Linha de orientação PC98	Linha de orientação PC99	PC Actual
Processador	Intel XScale 400 MHz	Intel PXA270 520MHz	ARM9 156 MHz	ARM9 104MHz	Pentium 200 MHz	Pentium 300 MHz	Processador 3GHz + ou equivalente
Memória	128 MB + 48 MB Flash ROM	64 MB + 64 MB FlashROM	64MB + 32 MB + 16MB FlashROM	3.4 MB	32 MB	64 MB	512 MB ou mais
Aceleração gráfica 2D/3D	Não	Não	Não	Não	Não	Sim	Sim
Resolução	240 x 320	480 x 640	208 x 320	176 x 208	1024x768 (Recomendada)	1280x1024 (Recomendada)	1600x1200 (Recomendada)
Profundidade de cor	16 bits	16 bits	24 bits	12 bits	32 bits	32 bits	32 bits

Tabela 1 – Comparação entre dispositivos móveis e computadores pessoais

Como se pode ver pela tabela, alguns dos dispositivos portáteis actuais estão ao nível dos PC's médios existentes há 4/5 anos atrás, que já tinham capacidades multimédia e de processamento 3D. Mas como é normal, as plataformas móveis encontram-se a anos-luz dos PC's actuais. No entanto, apesar de algumas plataformas móveis parecerem muito superiores aos primeiros PC's multimédia (mais MHz, mais memória), a verdade é que têm condicionantes que dificultam a tarefa de estar ao nível desses mesmos PC's.

Um dos condicionantes existentes é a capacidade de processamento. É verdade que um PDA pode atingir valores superiores em termos de ciclos de relógio (MHz) aos dos PC's multimédia mais antigos. No entanto, os PDA's normalmente reduzem os seus ciclos de relógio de modo a poupar energia, pois se os dispositivos corressem sempre à velocidade máxima o consumo de energia seria muito elevado. Assim a plataforma

portátil seria pouco portátil, visto que a bateria se extinguiria rapidamente. Outro facto que dá vantagem aos PC's multimédia mais antigo é o facto de terem sempre uma ou mais unidades de vírgula flutuante, o que dá vantagens enormes em termos de performance, principalmente no que concerne ao tratamento de gráficos tridimensionais. As plataformas móveis raramente suportam nativamente o uso de vírgula flutuante. Por fim, de alguns anos para cá todos os PC's vêm com placas aceleradoras 2D/3D, que suportam os API's mais populares, como o Direct3D e o OpenGL. Isso traz um ganho de performance enorme em termos de aplicações 3D. Os sistemas portáteis estão bastante limitados nesse sentido, pois só agora é que vai começar a surgir aceleração 3D, aceleração essa que se deverá cingir aos sistemas topo de gama durante algum tempo.

No entanto, existem atenuantes que aproximam a performance das plataformas móveis dos sistemas *desktop*. As capacidades de memória já são bastante boas, e superiores à dos primeiros PC's multimédia, embora também aqui se ponha o problema do consumo de energia, caso a memória tenha uso intensivo. Quanto há resolução de ecrã, é bastante inferior nas plataformas móveis à resolução normal dos *desktop*, o que minimiza a capacidade de processamento gráfico necessária à apresentação de gráficos elaborados.

Tendo em conta todos os factores, podemos concluir que já existem bastantes plataformas móveis com potencial para mostrar gráficos 3D, embora ainda não estejam optimizadas para o efeito. No entanto, já é o suficiente para o surgimento de bibliotecas 3D para a criação de gráficos 3D relativamente elaborados.

3 Bibliotecas Gráficas 3D para Plataformas Móveis

3.1 OpenGL ES

O API OpenGL ES é um novo standard da indústria que surgiu a 28 de Julho de 2003, data em que a sua primeira especificação (1.0) foi lançada. Este standard define uma variante e subconjunto do API OpenGL mais orientada para plataformas móveis. A principal diferença entre as implementações deste standard e as bibliotecas de *rendering* para sistemas *desktop* ou estações de trabalho é o foco na aritmética de vírgula fixa, ao invés da aritmética de vírgula flutuante, pois a vírgula fixa encontra um maior suporte nos processadores para plataformas móveis. Outra distinção entre este standard e uma implementação genérica do OpenGL ou de standards comparáveis, é o facto de as funcionalidades existentes serem implementadas do modo mais compacto possível e de forma a usar o menor número de recursos possível. Um dos aspectos interessantes que exemplifica a afirmação anterior é a omissão do glBegin/glEnd e da maioria dos pontos de entrada que são alojados entre as funções referidas.

O API OpenGL original disponibiliza implementações gratuitas da especificação para várias plataformas, estando abrangidos todos os principais sistemas operativos. O mesmo não se passa com o OpenGL ES, cuja responsabilidade pertence a outro grupo, designado por Khronos Group [7]. Este grupo apenas é responsável pela especificação, deixando para terceiros o desenvolvimento de implementações do API, não disponibilizando portanto implementações gratuitas do standard. No entanto, o grupo disponibiliza um *Adopters kit* que inclui uma implementação da Hybrid, mas que no entanto apenas é gratuito para membros do grupo, tendo os restantes interessados que pagar a quantia de 1.500 dólares.

De seguida serão referidas as características e outros aspectos da especificação OpenGL ES, bem como as implementações mais relevantes deste standard.

3.1.1 Características

O OpenGL ES é desenhado de modo a ser um API leve que sirva como standard da indústria, de modo a tomar vantagem do *hardware* de aceleração gráfica 3D que está a surgir nos dispositivos móveis. A versão 1.0 deste API, baseado no OpenGL 1.3, contém as seguintes características [8]:

Processamento Geométrico:

O processamento geométrico ocorre continuamente em tempo real, e é responsável em transformar coordenadas 3D de vértices em coordenadas 2D de ecrã. O OpenGL ES suporta as seguintes funcionalidades de processamento geométrico:

Vectores de vértices (*Vertex arrays*)
Pontos, linhas, triângulos
Pilha de matrizes (*Matrix stack*)
Ponto de vista (*Viewport*), alcance da profundidade (*DepthRange*)
Iluminação de vértices (*Vertex lighting*)
Modelo de sombreamento (*ShadeModel*)

Rasterização:

Rasterização é o processo de converter gráficos vectoriais (imagens descritas matematicamente como pontos conectados por linhas rectas) em imagens bidimensionais (padrões de pixeis) que podem ser guardadas e manipuladas como conjuntos de bits. Cada ponto da imagem contém informação como a cor e profundidade. Este processo consiste em dois passos: determinar quais são os pixeis do ecrã que serão ocupados pela primitiva e depois atribuir uma cor e um valor de profundidade a cada um desses pixeis. O OpenGL suporta as seguintes funcionalidades de rasterização:

Multisampling (opcional)
Pontos e pontos *anti-aliased*
Linhas e linhas *anti-aliased*
Face Culling de polígonos
Offset de polígonos – modo de enchimento

Mapeamento de texturas:

O mapeamento de texturas é um processo de design gráfico no qual uma superfície bidimensional (2D), designada por mapa de textura, é "embrulhado" num objecto tridimensional (3D). Assim, o objecto 3D mostra uma textura de superfície similar à da superfície 2D. O OpenGL ES suporta as seguintes funcionalidades de mapeamento de texturas:

Texturas 2D
Embrulhar (*Wrap*), repetir (*repeat*), colagem às arestas (*edge_clamp*)
Textura comprimida
TexSubImage, *CopyTexImage*
Multitexturas (*Multitexture*)
Formatos de pixeis e pacotes de pixeis RGBA, L, LA
Todos os filtros

Processamento de fragmentos:

O processamento de fragmentos descreve como é que os pixeis são sombreados. Anteriormente, o *pipeline* gráfico era fixo, permitindo poucas maneiras de se calcular o sombreamento de cada pixel. No entanto, com os avanços da tecnologia, o processamento de fragmentos é agora programável, podendo determinar o sombreamento de pixeis numa gama variada de maneiras. O OpenGL ES suporta as seguintes funcionalidades de processamento de fragmentos:

Nevoeiro (*Fog*)
Teste tesoura (*Scissor test*)
Teste alfa (*Alpha test*)
Teste de estêncil (*Stencil test*)
Teste de profundidade (*Depth test*)
Blending
Operação lógica (*Logic Op*)
Dither

Operações do *Framebuffer*/Outras:

As operações de *Framebuffer* incluem processos sofisticados como a mistura de alfa (*alpha blending*), sensação de desfasamento no movimento (*motion blur*) e *buffering* de profundidade (*depth-buffering*). O OpenGL suporta as seguintes funcionalidades de *Framebuffer*:

Clear
ReadPixels/Alpha test/Dither
Flush/Finish
Hint
Obter estados estáticos (constantes)

A versão actual (1.1) do OpenGL ES enfatiza a aceleração por *hardware* do API, mas é totalmente retro compatível com a versão 1.0. As novas características no OpenGL ES 1.1 providenciam funcionalidade aumentada, qualidade de imagem melhorada e optimizações para aumentar a performance enquanto reduzindo o uso de memória para poupar energia. Estas incluem:

- **OpenGL 1.5 usado como referência** – a especificação é definida relativa à especificação OpenGL 1.5.
- **Objectos em buffer** providenciam um mecanismo que os clientes podem alocar, inicializar e desenhar (render) a partir da memória. Os objectos em buffer podem ser usados para guardar vectores de vértices e dados de índex de elementos**.**

- **A geração automática de *mipmaps*** pode aliviar a aplicação do processo de gerar *mip-levels*. Implementações de *hardware* podem potencialmente acelerar a geração de *mip-levels*, especialmente para texturas vídeo ou quando desenha (render) para textura. Uma textura é considerada incompleta no OpenGL ES quando um conjunto de vectores de *mipmap* não é especificado com o mesmo tipo. A verificação de integridade é feita quando uma dada textura é usada para desenhar (*render*) geometria.

- **Processamento de texturas melhorado**, incluindo um mínimo de duas multi-texturas e uma funcionalidade de combinação de texturas para efeitos como o mapeamento de superfícies (*bump mapping*) e iluminação por pixel. Todos os ambientes de textura do OpenGL 1.5 são suportados, com a excepção da barra transversal de texturas (*texture crossbar*).

- A **funcionalidade de *vertex skinning*** permite uma animação mais macia de figuras complexas e geometrias, usando a extensão oes_matix_palette. A extensão permite ao OpenGL ES suportar uma palete de matrizes. A palete de matrizes (*matrix palette*) define um conjunto de matrizes que podem ser usadas para transformar um vértice. A palete de matrizes não é parte da pilha de vista de modelo (*model view stack*).

- Os **planos de clip definidos por utilizador** (*User-defined clip planes*) permitem um eficiente e antecipado *culling* de polígonos não visíveis, aumentando a performance e poupando energia.

- ***Sprites* de pontos (*point sprites*) e vectores de *sprites* de pontos** providenciam um método para desenhar partículas usando pontos em vez de *quads*. Isto permite efeitos de partículas realísticos e eficientes. A extensão de *sprites* de pontos também permite a uma aplicação especificar coordenadas de textura específicas que são interpoladas ao longo do ponto, em vez da mesma coordenada de textura usada pelos tradicionais GL points (pontos GL). A extensão Point Size Array permite que um vector de tamanhos de pontos em vez de um tamanho fixo seja inserido na criação do ponto, o que providencia flexibilidade para as aplicações criarem efeitos de partículas.

- A **devolução de estados dinâmicos e estáticos** (*static and dynamic state queries*) é suportada para estados dinâmicos e estáticos especificamente suportados no perfil. Os GL *state queries* suportados podem ser categorizados em simples *queries*, *queries* enumerados, *queries* de texturas, *queries* de apontadores e *strings*, e *queries* de objectos de buffer. Isto permite ao OpenGL ES ser usado em sofisticados ambientes de desenvolvimento de *software* com camadas.

- ***Draw Texture*** (desenho de textura) define um mecanismo para desenhar rectângulos de pixeis a partir de uma ou mais texturas para uma região rectangular do ecrã. Esta capacidade é útil para fazer um rápido *rendering* de pinturas de fundo, *bitmapped font glyphs*, e elementos de emolduramento 2D em jogos.

- **Nova adições nucleares e extensões de perfis** para os perfis Common e Common-Lite adicionam subconjuntos às coordenadas de *byte* OES, OES fixed – point, OES precisão simples e matriz OES, que passam assim a ter extensões específicas no ES como adições ao núcleo; OES – formato de leitura, OES *compressed palleted texture*, OES *point size array* e OES *point sprite* como extensões de perfil requeridas; e OES *matrix palette* e OES *draw texture* como extensões de perfil opcionais.

3.1.2 OpenGL ES *framework*

3.1.2.1 Perfis

A especificação OpenGL ES inclui a definição de vários perfis. Cada perfil é um subconjunto da especificação do OpenGL 1.5 mais algumas extensões específicas do OpenGL ES. Os perfis do OpenGL ES fazem parte de uma mais vasta família de programação de interfaces de aplicações derivados do OpenGL. Sendo assim, os perfis partilham um *pipeline* de processamento similar, estrutura de comandos, e o mesmo *name space* OpenGL. Quando necessário, são criadas extensões para aumentar a funcionalidade actual do OpenGL 1.5. A extensões específicas do OpenGL ES têm um papel nos perfis do OpenGL ES similar ao papel que foi desempenhado pelas extensões OpenGL ARB relativas à especificação OpenGL. As extensões específicas ao OpenGL ES são ou percursores de funcionalidade destinados à inclusão em futuras revisões de núcleo, ou a formalização de funcionalidades importantes, mas que não são *mainstream*. Cada definição de perfil implica um ficheiro de header distinto e uma *link/runtime library* definindo os comandos e os *tokens* que estão no perfil. Para simplificar a manutenção um único *header* pode ser definido com as apropriadas directivas de pré processamento para controlar a visibilidade dos *tokens* e dos protótipos de comando. Em tempo de execução, uma aplicação pode determinar que perfil está a correr usando o OpenGL *version string query*.

Actualmente a especificação consiste num total de duas definições de perfil: o perfil Common (comum), e o perfil Safety Critical (segurança crítica).

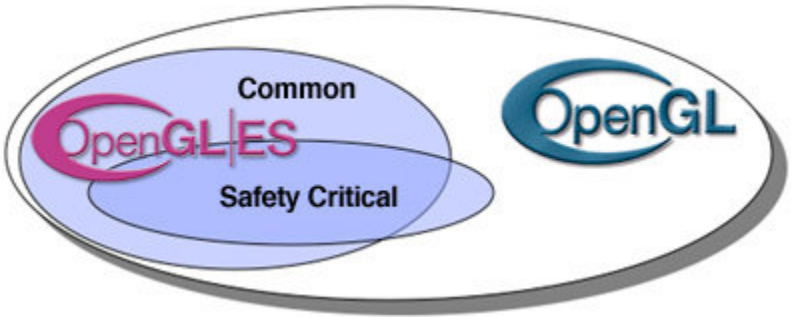

Figura 1 – Perfis do OpenGL ES

- O perfil Common é vocacionado para o entretenimento e aparelhos relacionados, como os telefones móveis, PDA's, *set-top boxes*, consolas de jogos, etc. Está virado para uma mais vasta gama de mercado, incluído o suporte para plataformas com grande capacidade de variação. Existe uma variação deste perfil designada por Common-Lite, cuja única diferença é não suportar vírgula flutuante:

    - Suporte total de funções 3D com mapeamento de textura
    - Boa plataforma de jogos
    - Implementável em telefones móveis

- O perfil Safety Critical é vocacionado para aplicações para consumo e industriais onde a fiabilidade e certeza são características fundamentais.

    - Funções 3D mínimas para facilitar as certificações de segurança
    - Usado na aviação e em ecrãs de automóveis

3.1.2.2 Conformidade

As especificações são revistas numa base anual. Para etiquetar uma implementação como sendo conforme ao OpenGL, é necessário que esta passe por uma série de testes de conformidade.

3.1.2.3 Extensões

As implementações OpenGL podem incluir extensões que adicionam novas funcionalidades à implementação. Um perfil OpenGL ES consiste em duas partes: um subconjunto do *pipeline* OpenGL completo, e algumas funcionalidades estendidas que são retiradas de um conjunto de extensões específicas do OpenGL ES. Cada extensão é feita de modo a ser compatível com o subconjunto de comandos do perfil e a ser adicionada ao mesmo, tanto como uma adição ao núcleo, ou como uma extensão ao perfil. As adições ao núcleo diferem das extensões ao perfil no facto de os seus comandos e tokens não incluírem sufixos de extensão no seu nome. As extensões de perfil são adicionalmente divididas em extensões requeridas (obrigatórias) e opcionais. As extensões requeridas têm de ser obrigatoriamente implementadas numa aplicação conformável, enquanto a implementação de extensões opcionais é deixada ao critério do implementador.

3.1.2.4 Plataform Interface Layer (camada de interface com plataforma) – EGL

O OpenGL ES também inclui uma especificação da camada comum de interface com a plataforma, designada por EGL. Esta camada é independente da plataforma e pode opcionalmente ser incluída como parte da distribuição OpenGL do vendedor. A ligação às plataformas também tem associado um teste de conformidade. Alternativamente, o vendedor pode optar por definir a sua própria camada de envolvimento específica à plataforma.

3.1.3 Vantagens do OpenGL ES

Standard da indústria e isento de Royalties

Qualquer pessoa pode fazer o *download* da especificação OpenGL ES e implementar produtos baseados na mesma. O estandardizado alto nível de abstração que oferece significa que os fabricantes se possam concentrar mais no conteúdo e menos em detalhes de código e de plataforma.

Baixo consumo de energia e uso de memória

As plataformas de aplicação variam muito, podendo ser aplicado desde PDA's com 400Mhz e 64MB de RAM a telefones moves com 50Mhz e 1MB de RAM. O OpenGL ES é desenhado para acomodar as diferenças entre plataformas, tendo requisitos mínimos de armazenamento de dados, tráfego de instruções/dados minimizado, e é compatível tanto com inteiros e vírgulas flutuantes. Para os utilizadores isto significa binários mais pequenos para *download* e menos espaço ocupado no dispositivo.

Transição transparente entre o render por *software* e por *hardware*

Apesar da especificação OpenGL ES definir um *pipeline* de processamento gráfico particular, chamadas individuais podem ser executadas em *hardware* dedicado, correr como sub rotinas de *software* no processador do sistema, ou ser implementado como uma combinação tanto de *hardware* dedicado como rotinas de *software*. Isto significa que os fabricantes de *software* podem desenvolver um motor 3D compatível com *software* hoje, e mais tarde fazer uma transição sem grande esforço para usar a aceleração por *hardware* do OpenGL ES na próxima geração de dispositivos.

Extensível e em constante evolução

O OpenGL ES permite que inovações de *hardware* sejam acessíveis através de API usando o mecanismo de extensão do OpenGL e que a API seja facilmente actualizada. À medida que uma extensão vai tendo mais aceitação, passa a ser considerada a sua inclusão no núcleo do standard OpenGL ES. Este processo permite ao OpenGL ES evoluir de uma maneira controlada mas inovadora.

Bem documentado

Sendo baseado no OpenGL, existem inúmeros livros e uma grande quantidade de código de exemplo, fazendo com que a informação acerca do OpenGL ES seja barata e fácil de encontrar. Com a introdução do OpenGL ES, um fabricante pode na teoria escrever basicamente o mesmo código desde telefones móveis a super computadores.

3.1.4 Implementações

O OpenGL ES por si só não permite o desenvolvimento de aplicações, visto que é apenas uma especificação, não disponibilizando portanto binários de modo a que possa ser implementado nas aplicações finais. No entanto, existem já algumas implementações que estão em conformidade com a especificação e permitem, portanto, o seu uso nas mais variadas plataformas. É sobre algumas dessas implementações que nos debruçaremos em seguida.

3.1.4.1 Hybrid Gerbera

A Hybrid é uma empresa finlandesa que é especializada em tecnologia gráfica para dispositivos sem fios e para a Internet. A empresa atingiu alguma notoriedade ao criar a primeira implementação da especificação OpenGL ES 1.0, e desde então tem sido uma referência no sector dos gráficos 3D para plataformas móveis. Entretanto a empresa tornou-se um membro activo do Khronos Group, grupo responsável pela especificação OpenGL ES, e de outros grupos de estandardização relacionados.

Na sua lista de clientes da Hybrid estão vários nomes importantes da indústria:

- ATI: fabricante de chipsets gráficos líder de mercado e que se prepara para lançar chipsets para plataformas móveis.
- Bitboys: fabricante de chipsets gráficos para plataformas móveis.
- Discreet: criadores de ferramentas gráficas, como a 3ds Max.
- Ericsson: fabricante de telemóveis de renome.
- Fathammer: pioneiros no desenvolvimento de jogos para plataformas móveis, e fabricantes dos kits de desenvolvimento X-Forge que permitem criar jogos 3D para plataformas móveis.
- Futuremark: empresa especializada na criação de *benchmarks* (PC Mark 04, 3D Mark 04), incluindo um especializado *smartphones*[1], designado por SPMark 04.
- Sony Online Entertainment: divisão da Sony especializada em jogos online.
- Symbian: sistema operativo para plataformas móveis, que passará a contar com a implementação de OpenGL ES da Hybrid de raiz a partir da versão 8.0.
- Synergenix: criadores da plataforma de desenvolvido de jogos para plataformas móveis mophun.

Um dos principais produtos desta empresa é a sua framework OpenGL ES, que tem no seu núcleo a implementação do API OpenGL ES, designada por Gerbera. É sobre esta implementação que nos debruçaremos nas próximas páginas.

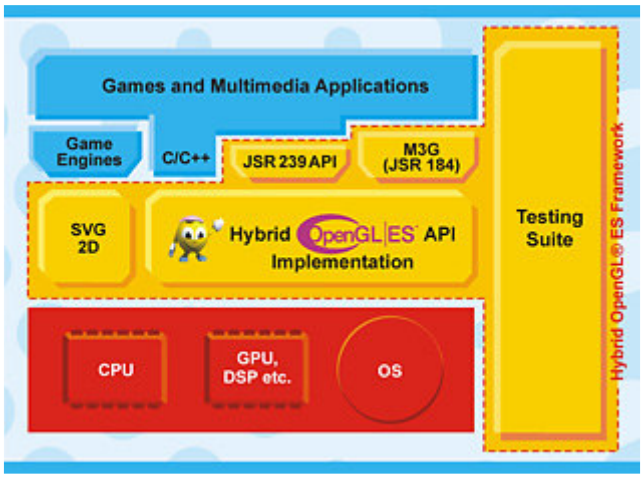

Figura 2 – Framework OpenGL ES da Hybrid

O objectivo principal do Gerbera é providenciar um *renderer* 3D estandardizado para todas as plataformas móveis. Para cumprir esse objectivo a plataforma tenta ser robusta (um dispositivo móvel não pode bloquear), tem boa performance e implementa todas as características do OpenGL ES, e é modelar e portátil, adaptando-se à aceleração por *hardware*.

Em adição ao componente para as plataformas móveis, a plataforma inclui ainda uma ferramenta de testes extensiva, um SDK para desenvolvimento de aplicações em PC, e API's adicionais para fins de *debugging* e *profiling*. Todos juntos, estes componentes formam a plataforma OpenGL ES da Hybrid.

De referir ainda que nas plataformas onde não há suporte de *hardware* para vírgula flutuante, o Gerbera opera internamente usando aritmética baseada em inteiros. A exactidão, precisão e limites são aproximadamente aqueles definidos pelo IEEE para as vírgulas flutuantes de precisão simples, enquanto que a perda de performance é mínima. Nas plataformas onde existe suporte de *hardware* para vírgulas flutuantes, elas poderão ser usadas.

3.1.4.1.1 Arquitectura

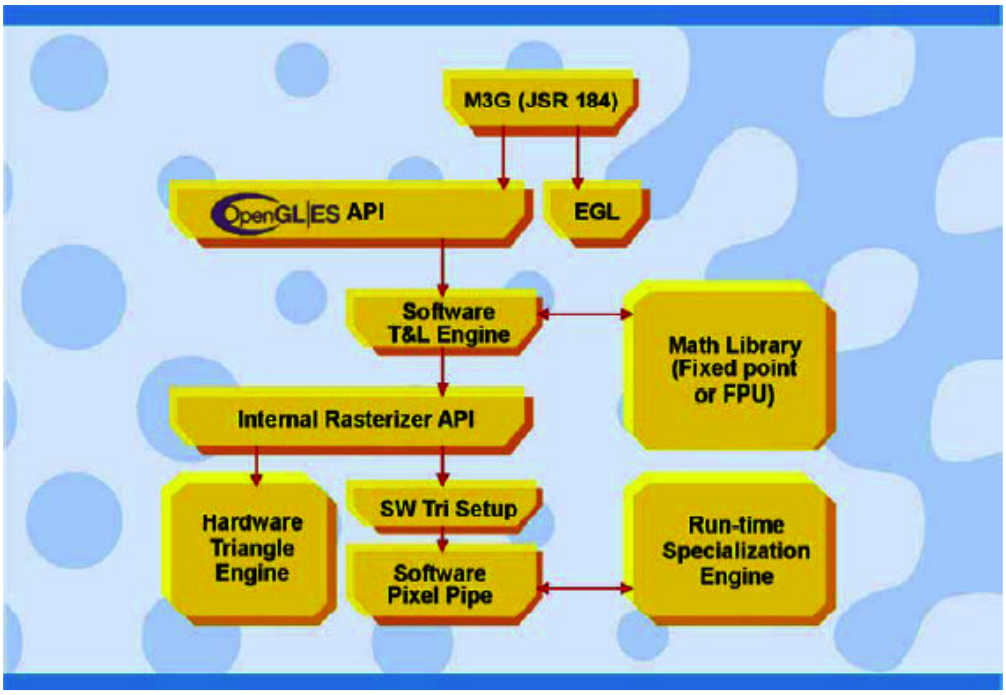

Figura 3 – Componentes principais da framework [G] Hybrid Gerbera

Os principais componentes do Gerbera são o seu motor de *Transform and Lighting* (transformação e iluminação), rasterizador de triângulos, biblioteca matemática e motor de especialização em tempo de execução (*run-time specialization engine*). O sistema é escrito em C portátil. Para plataformas ARM, várias porções foram especificamente optimizadas para em assembler ARM.

O motor de transformação e iluminação usa uma biblioteca matemática especializada. O principal objectivo dessa biblioteca matemática é providenciar a escala e precisão requeridos pela especificação OpenGL ES. A biblioteca matemática foi desenhada com o processamento de inteiros com vírgula fixa em mente, mas pode também suportar uma unidade de vírgula flutuante (FPU), se presente.

Um API interno de rasterização separa a rasterização de triângulos do resto do sistema. Para plataformas sem nenhum *hardware* 3D, uma solução inteiramente baseada em *software* é providenciada. Os rasterizadores de *hardware* 3D podem ser suportados através do API interno de rasterização.

O código da parte mais crítica em termos de tempo do *renderer*, o *pipeline* de pixeis, é gerado no momento usando um compilador JIT [G] desenvolvido. Esta funcionalidade reside num motor de especialização em tempo de execução (*run-time specialization engine*). O Gerbera avalia o estado do *pipeline* de pixeis activo e cria rotinas usando uma linguagem intermédia. Um compilador separado faz então várias

optimizações no código intermediário, por exemplo, eliminação de código inútil, propagação constante, alocamento no registo, calendarização de instruções e outras optimizações. Finalmente, o código optimizado do *pipeline* de pixeis é processado por um *back-end* que traduz o código em código binário nativo.

O *back-end* também faz optimizações específicas ao processador. Os programas gerados são colocados numa cache de código para que não seja necessário recompilá-los quando re-usados pela aplicação. O tamanho aproximado do motor de especialização em tempo de execução é de 20K, e os tempos de compilação do *pipeline* são insignificantes.

A camada do API OpenGL ES para o Gerbera providencia funcionalidade não suportada pelo próprio núcleo, por exemplo, pilhas de matrizes, relatórios de erros e manuseamento de texturas. A especificação OpenGL ES permite que as implementações variem ligeiramente nas características suportadas, exactidão de *rendering* e qualidade. O Gerbera consistentemente opta pelas alternativas de maior qualidade, como as rápidas transformações de qualidade de vírgula flutuante do IEEE, suporta todas as características opcionais e ainda providencia performance suficiente para conteúdos 3D ricos.

3.1.4.1.2 Aceleração gráfica

A aceleração gráfica por *hardware* não é obrigatoriamente necessária. Contudo, pode ser usada para aumentar muito a performance.

Existem vários esquemas possíveis para a aceleração por *hardware*. Estes são alguns dos esquemas típicos:

- Operações de aceleração 2D, como limpezas de ecrã podem ser totalmente utilizadas pelo Gerbera.
- O Gerbera pode ser optimizado para tomar vantagem de conjuntos de instruções multimédia do processador (por exemplo, MMX e 3Dnow).
- O Gerbera pode tomar partido de processadores extra como DSP, núcleos de processadores adicionais e co-processadores multimédia.
- Rasterizadores de *hardware* 3D – O Gerbera pode redireccionar a rasterização de triângulos para o *hardware*.
- Aceleração completa por *hardware* 3D – A *framework* do Gerbera pode ser usada para testar e verificar o sistema completo, incluindo teste e verificação de *drivers* OpenGL ES de outras fontes.

Em todos os casos em que a aceleração por *hardware* é necessária, o processo começa sempre com a testada e conformável implementação por *software*. As funcionalidades relevantes são depois substituídas pelo *hardware*, baseando-se no *hardware* utilizado. Deste modo existe um suporte OpenGL ES conformável desde o início. Isto é significantemente mais fácil do que escrever uma nova funcionalidade para o *driver* do zero.

3.1.4.1.3 Sistemas operativos e processadores suportados

A *framework* do API OpenGL ES da Hybrid está disponível para os seguintes sistemas operativos e processadores:

Sistemas Operativos:

- Symbian 6.0
- Symbian 7.0
- Nokia Series 60
- UIQ (2.1)
- Windows Mobile 2002 para PocketPC
- Windows Mobile 2003 para PocketPC
- Windows Mobile 2003 para SmartPhone
- Intent (1.5)
- Windows 2000
- Windows XP

Também são suportados mais alguns sistemas operativos proprietários não divulgados.

Processadores:

- ARM7
- ARM9
- Intel XScale
- TI OMAP
- SH3-DSP
- x86

3.1.4.1.4 Processo de integração

O Gerbera pode ainda ser implementado num dispositivo móvel alvo, ainda não suportado, quer por ser novo, quer por ser um sistema proprietário. Para isso são seguidos os seguintes passos:

1. Avaliação do dispositivo alvo que resulta num relatório de integração que inclui detalhes e estimativas de trabalho para:
    a. A possível utilização de *hardware* gráfico
    b. O componente EGL dependendo do sistema operativo
    c. A integração em cadeia no dispositivo móvel

2. Uma versão protótipo – uma versão em código C simples com todas as características suportadas a correr no dispositivo alvo.
3. A versão final, testada e optimizada, pronta a ser distribuída.

3.1.4.1.5 Dados técnicos

Para finalizar, a seguinte tabela contém os dados técnicos da *framework* sumariados:

Perfis suportados	OpenGL ES 1.0 Common and Common Lite, EGL 1.0
Processadores	x86, ARM 7/9/9E, SH3-DSP, Intel XScale, TI OMAP
Sistemas Operativos	Win32, Symbian, WinCE, vários proprietários
Compiladores	MSVC, MSEVC, GCC, ADS
Resolução Máxima	2048x2048
Formatos de *Buffer* de Cores	8/16/32 bpp arbitrários + *dithering* de formatos de ecrã arbitrários
Destination Alpha	Suportado
Formatos do *Buffer* de Profundidade	8/16/32 bpp
Formatos do *Stencil Buffer*	8 bpp
Correcção de Perspectiva	Todos os gradientes, exactidão total
Unidades de Texturas	2
Tamanho do código	100K
Uso da *Heap*	80K
Uso da *Stack*	1K
Extensões OpenGL ES suportadas	OES_byte_coordinates OES_compressed_paletted_texture OES_fixed_point OES_query_matrix OES_read_format OES_single_precision
Ferramentas adicionais	Emulador para PC, *benchmarks*, pacote de testes automatizado

Tabela 2 – Dados técnicos da framework OpenGL ES da Hybrid (Gerbera)

3.1.4.2 Vincent

O Vincent é o primeiro projecto Open Source [G] a apresentar uma implementação do OpenGL ES 1.0 que passou os testes de conformidade, concentrando-se numa primeira fase em implementar o perfil Common-Lite. Este perfil é uma variação simplificada do perfil Common, que providencia variantes de vírgula fixa para todas as chamadas típicas de uma biblioteca gráfica que tradicionalmente empregam vírgula flutuante.

As plataformas alvo iniciais deste projecto são os dispositivos com o sistema operativo Windows Mobile, mais propriamente os PocketPCs e os Smartphones que usam o processador Intel XScale PA2xx. A implementação nestes sistemas torna-se facilitada visto que os formatos numéricos e as primitivas de vectores e matrizes requeridas para a implementação estão disponíveis directamente através da Graphics Performance Library, disponibilizada pela Intel.

Na tabela seguinte encontra-se uma tabela com as datas e objectivos do projecto.

Data	Versão	Objectivo	Descrição
28/12/2003	0.2	Conceito	Incluiu basicamente a implementação da referência do rasterizador
25/03/2004	0.7ª	Arquitectura	Lançamento inicial da geração de código em *runtime*
18/05/2004	0.75	Lançamento da Beta	Conclusão interna bem sucedida dos testes de conformidade relativos à funcionalidade do núcleo.
26/07/2004	0.8	**Correcção**	Conclusão interna bem sucedida dos todos os testes de conformidade. [5]
31/08/2004	0.9	Performance	Versão optimizada e afinada
??/??/2004	1.0	Validação	Submissão aos testes de conformidade e aceitação oficial.
As datas a itálico são datas futuras estimadas			

Tabela 3 – Marcos do projecto

3.1.4.2.1 Arquitectura

Estrutura da biblioteca

A biblioteca de *rendering* 3D é distribuída como uma DLL do Windows designada GLES_CL.DLL de acordo com a especificação do API OpenGL ES. A interface pública desta DLL é definido nos *header files* GLES/egl.h, GLES/gl.h e GLES/egltypes.h. Um *import* da livraria GLES_CL.LIB pode ser usado para fazer o link do executável à DLL.

Apesar da API OpenGL ES ser especificada como sendo uma API de linguagem C, este implementação foi implementada usando o C++. Sendo assim, uma série de funções correspondentes é fornecida que mapeia a API de linguagem C em chamadas da subjacente implementação C++.

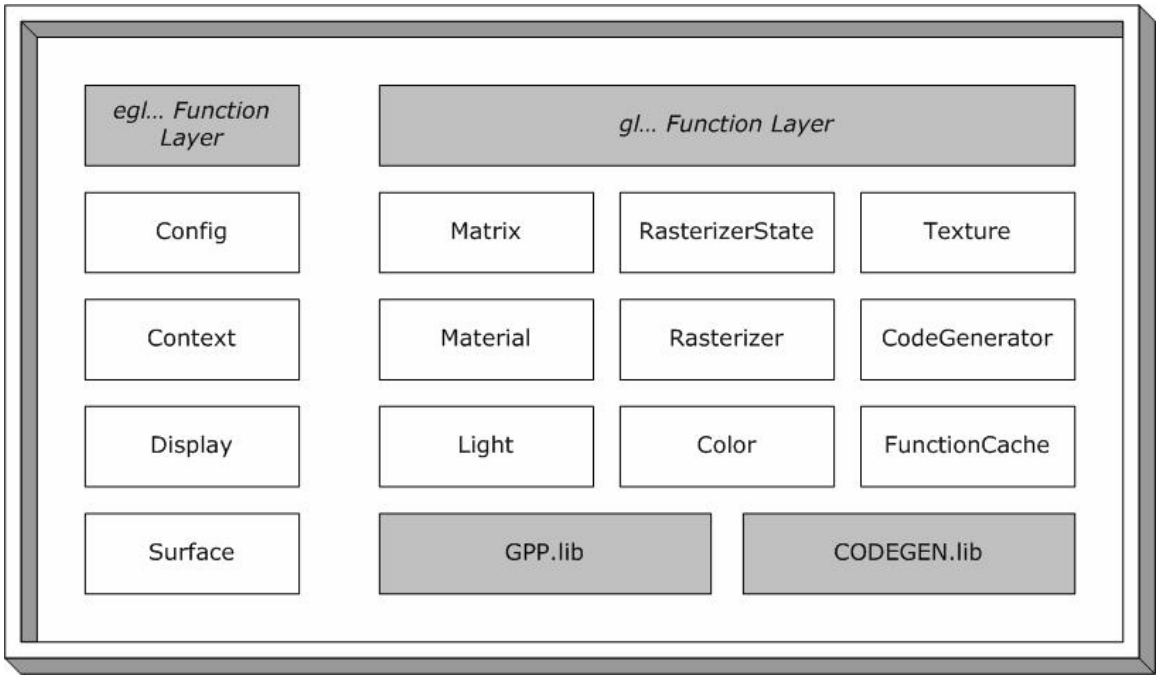

Figura 4 – Estrutura do código da implementação Vicent 3D

Geração de Código em Runtime

Um diferenciador chave deste projecto de outros esforços open source similares, é o uso de geração de código em tempo de execução (*Runtime*). Esta aproximação é usada para criar versões optimizadas das funções de conversão do varrimento (*scan conversion functions*) em tempo de execução baseadas nas definições actuais de rasterização.

Cada chamada do glDrawArrays ou glDrawElements causa a re-validação por parte do rasterizador e inicializa as suas definições internas antes da correspondente sequência de primitivas geométricas ser enviada para o *pipeline* gráfico. Como parte desta re-inicialização, o rasterizador emprega um gerador de código para compilar uma função optimizada para a conversão *scanline* usando um compilador JIT que é parte da biblioteca. Uma cache de funções captura as versões mais recentemente usadas das funções *scanline* e assegura que a recompilação do código das funções é mantida ao mínimo. Como input ao compilador JIT, o gerador de código usa uma variante da

linguagem intermédia triVM de Johnson's. Esta linguagem intermédia foi especialmente desenhada para optimização de código nas plataformas com arquitectura ARM.

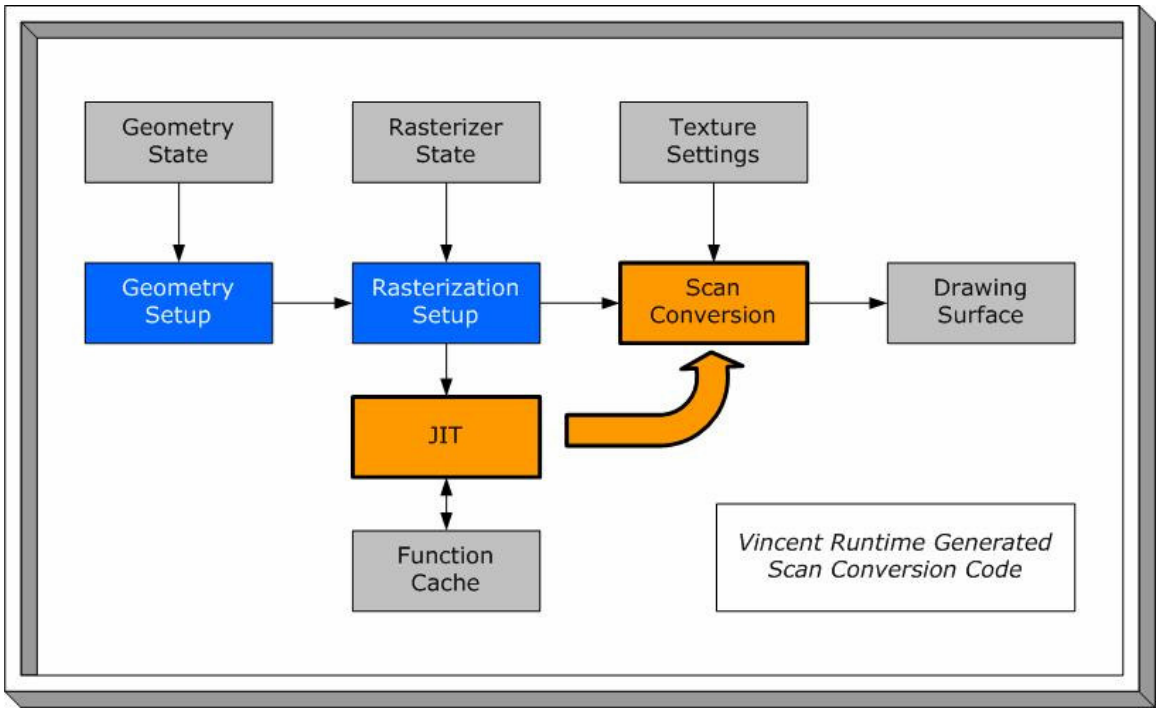

Figura 5 – Geração de código em tempo real do Vincent 3D

3.1.4.2.2 Guia de desenvolvimento

O desenvolvimento de aplicações usando o Vincent 3D pode ser feito de modo gratuito, visto que as ferramentas essenciais estão disponíveis gratuitamente. Como tal, de seguida está um pequeno guia de desenvolvimento criado pela equipa do Vincent 3D. Com este guia pretende-se que o programador configure correctamente as ferramentas necessárias de modo a começar o desenvolvimento propriamente dito.

Pré-requisitos

De modo a usar esta biblioteca para desenvolver um projecto, é necessário a instalação do seguinte *software* no PC de desenvolvimento:

- SDK for Pocket PC 2003 e/ou SDK for Smartphones 2003
- eMbedded Visual C++ 4.0 SP3
- (Opcional) Intel C++ Compiler

Deste *software* só o Intel C++ Compiler não é disponibilizado gratuitamente, mas não é necessária sua utilização, embora prometa uma maior performance.

Para além do *software* referido, é necessário alguma experiência na utilização do API OpenGL e na programação em C/C++ usando o Win32 API.

Em que consiste esta biblioteca?

A distribuição consiste nos seguintes componentes:

- Um conjunto de *header files* GLES/egl.h, GLES/gl.h e GLES/egltypes.h.
- Uma *import library* GLES_CL.LIB.
- Uma *runtime library* GLES_CL.DLL.

De modo a compilar um programa C ou C++ usando esta biblioteca, é necessário ter as *header files* incluídas no *path* de procura. Para fazer o *link* a um executável ou da DLL desta biblioteca é necessário que a *import library* esteja nos directórios de *input* do *linker*. Para executar o programa, é necessário que a *runtime library* esteja no dispositivo móvel ou dentro da imagem do emulador do dispositivo.

Configurar o eMbedded Visual C++

Para usar a biblioteca 3D, é necessário incorporar as *include files* e a *import library* no directório de procura do ambiente de desenvolvimento utilizado. A maneira mais fácil de o fazer é modificar as definições de directório usando o comando "Tools > Options… ". Na caixa seguinte, escolha o *tab* "Directories". Resta então incluir as localizações dos *include files (headers)* e *import library* nas secções correspondentes.

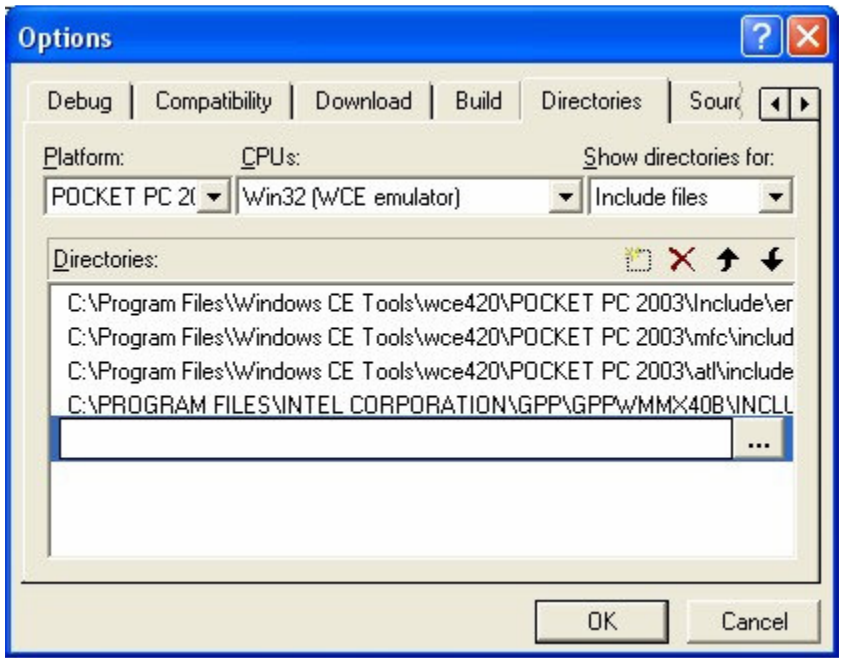

Figura 6 – Inclusão das *include* e *library files* no projecto

Definições do Compilador

Depois de adicionados os directórios de *include* às definições do ambiente de desenvolvimento, não é necessário qualquer ajuste adicional às definições do compilador. Para usar as funcionalidades da biblioteca 3D podem ser usadas adicionando as seguintes linhas ao código C/C++:

```
#include <GLES/egl.h>
#include <GLES/gl.h>
```

Definições do *Linker*

Para criar um executável do Windows ou uma DLL, é necessário incluir a *import library* da biblioteca 3D no *input* do *linker*. Isso pode ser feito adicionando uma referência ao ficheiro GLES_CL.LIB nas definições acessíveis através do "Project > Settings…".

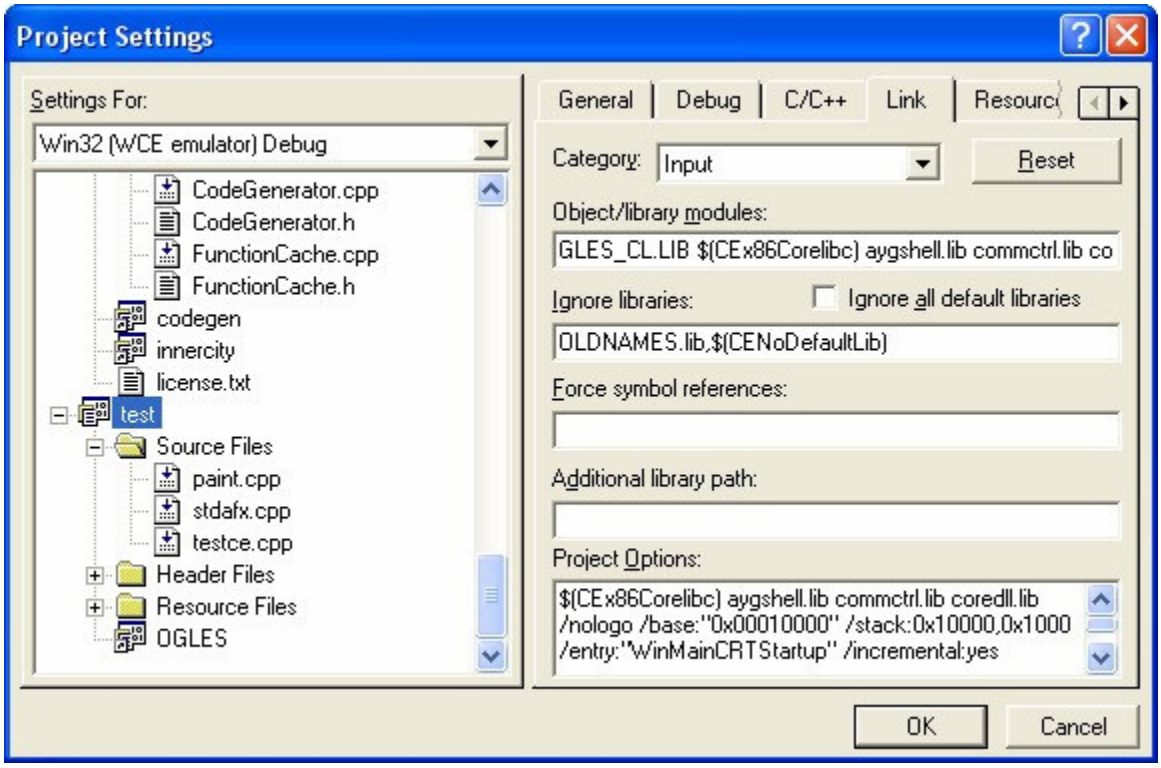

Figura 7 – Inclusão do ficheiro GLES_CL.LIB nas definições do *Linker*

Configuração em Tempo de Execução (Runtime)

Para executar o programa final usando a implementação Vincent 3D, é necessário incluir a DLL GLES_CL.DLL na plataforma móvel ou na imagem do emulador. Isso pode ser feito copiando a DLL ou para o directório onde está o executável, ou para a pasta \Windows no dispositivo. Ter em atenção que a versão da DLL a ser usada deve corresponder ao processador do dispositivo a utilizar.

3.1.4.2.3 Sistemas operativos e processadores suportados

A implementação OpenGL ES Vincent 3D está disponível para os seguintes sistemas operativos e processadores

Sistemas Operativos:

- Windows Mobile 2003 para PocketPC
- Windows Mobile 2003 para SmartPhone
- Windows 2000
- Windows XP

Processadores:

- Intel StrongARM
- Intel XScale
- TI OMAP
- x86

Como alvos seguintes estão definidos os sistemas operativos Symbian.

3.2 Mobile 3D Graphics (JSR 184)

As aplicações não Java são normalmente compiladas no chamado código nativo que é normalmente executado directamente pelo *hardware*. As aplicações Java, por outro lado, são expressas em *bytecode* independente do *hardware* que é executado por uma máquina virtual. Isto impõe uma penalização de performance variável, que é tolerável em aplicações normais, mas proibitiva para gráficos em tempo real.

Como se pode ver no gráfico que se segue (Figura 8 – Diferença de performance entre o código nativo e máquinas virtuais de Java), o código Java pode correr até vinte vezes mais lentamente que o código nativo. Esta situação vai melhorando com aceleradores de *hardware* Java, assim como novas máquinas virtuais que compilam *bytecode* em código nativo, mas mesmo nesses casos a barreira de performance mantém-se.

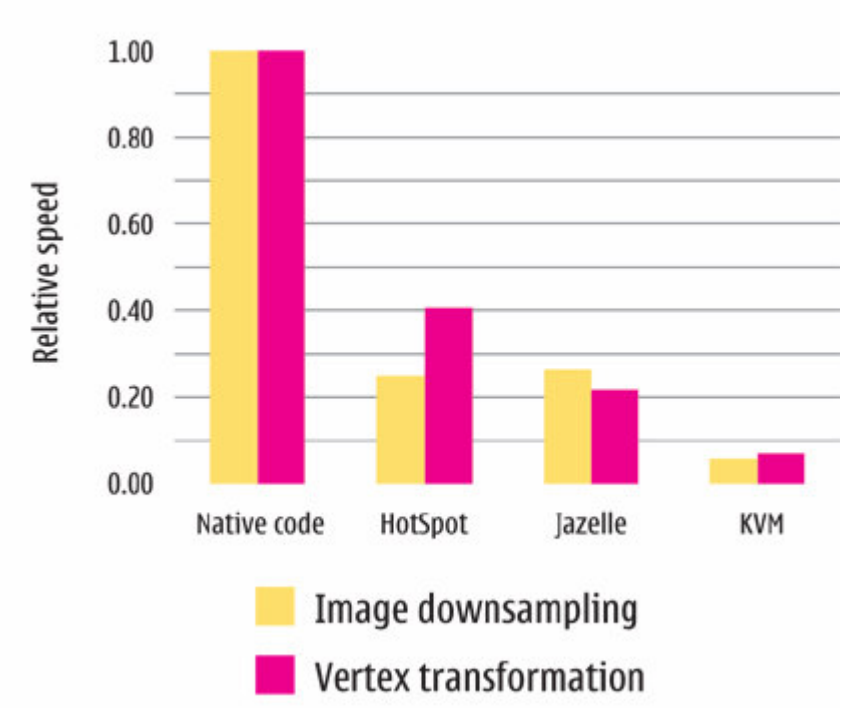

Figura 8 – Diferença de performance entre o código nativo e máquinas virtuais de Java

Como já foi dito e se vê na figura, o código nativo corre até vinte vezes mais rápido que na máquina virtual de Java predominante, a KVM. A máquina virtual de próxima geração HotSpot e o acelerador de *hardware* Jazelle são muito mais rápidos que a KVM, mas ainda perdem para o código nativo num factor de três para quatro no processamento gráfico.

Como os motores gráficos eficientes não podem ser escritos em Java, é necessário que um seja implementado em código nativo (ou mesmo em *hardware* dedicado) e que

seja disponibilizado como parte da plataforma Java. É esta visão que levou à criação do Mobile 3D Graphics API (M3G).

O JSR 184 entrou na fase de lançamento final em Dezembro de 2003. O M3G é um pacote opcional do J2ME (javax.microedition.m3g) desenhado para providenciar um eficiente API gráfico 3D adequado à plataforma J2Me, e, em particular aos perfis Conected Limited Device Configuration (CLDC) e Mobile Information Device Profile (MIDP). Estes perfis são utilizados por plataformas com poder de processamento limitado e nenhum suporte de *hardware* para gráficos 3D e matemática de vírgula flutuante. No entanto, a API também deve ser escalável até dispositivos de topo de gama com ecrãs a cores, um DSP (*Digital Signal Processor*), uma unidade de vírgula flutuante, e até *hardware* gráfico 3D especializado. O API pode assim suportar características que não são estritamente necessárias em plataformas de baixa gama. O M3G tem como alvo uma variedade de aplicações (incluindo jogos, visualização de mapas, interfaces com o utilizador, visualização de produtos e mensagens animadas).

O M3G inclui API de baixo nível, referida como modo imediato, que é um subconjunto do OpenGL. Também inclui um API de mais alto nível, designado por modo retido (*scene-graph* – grafo de cena), construído em cima do API de modo imediato, assim como importadores de elementos tal como *meshes*, texturas, animações, e hierarquias de cenas. A implementação de referência tem o tamanho de 150 KB, incluindo o motor gráfico nativo, as classes Java, e ferramentas de importação de conteúdo.

De referir ainda que existem vários standards JSR (Java Specification Requests) relacionados com esta API. São eles:

JSR-134

Um API desenhado e optimizado especialmente para jogos em plataformas com baixos recursos. Apesar de abordar o 3D até certo nível, é específico para os jogos. O JSR 184 (M3G) é desenhado para ser um standard 3D de uso geral.

JSR-135

Este API é desenhado para controlar o *playback* de uma variedade de tipos de media em dispositivos móveis. Embora não aborde os gráficos 3D, pode ser usado em conjunção com o M3G para controlar a visualização de conteúdo 3D.

JSR-148

Esta especificação aborda a as imagens volumétricas e a matemática vectorial 3D. Uma diferença é que o M3G representa a data como polígonos, e não como dados volumétricos. Entretanto, o pacote de matemática vectorial está relacionado com o M3G.

JSR-239

Este JSR será abordado com mais pormenor neste relatório. Consiste em criar ligações directas ao OpenGL ES a partir do Java, pelo que quando for concluído será uma alternativa real ao M3G.

JSR-912

Este é o standard Java 3D já existente. No entanto, não está optimizado para ambientes de baixos recursos como as plataformas móveis.

3.2.1 Características

A especificação M3G foi baseada em requerimento, sumariados em baixo, que foram acordados pelo grupo de especialista que desenvolveu a especificação. Cada requerimento está sumariado de seguida:

- O API tem de suportar o modo retido de acesso (ou seja, um grafo de cena).
- O API tem de suporto modo imediato de acesso, com características semelhantes às do OpenGL.
- O API tem de suportar a mistura e a combinação entre o modo imediato e o modo retido.
- O API não pode incluir partes opcionais (ou seja, todos os métodos têm que ser implementados).
- O API tem de ter importadores para *meshes*, texturas, grafos de cenas inteiros, etc.
- O API tem de ser eficientemente implementado no cimo do OpenGL ES.
- O API tem de usar tipo de dados *float* nativo do Java, e não introduzir um tipo alterado.
- O API tem de ser implementado eficientemente sem o recurso a *hardware* de vírgula flutuante.
- O API deve ser implementado em cerca de 150 KB num terminal móvel real.
- O API tem de ser estruturado para que a *garbage collection* seja minimizada.
- O API tem de ser interoperável com API's Java relacionados, especialmente com o MIDP.

3.2.2 Arquitectura – Classes

O M3G é um API baseado em objectos, e como tal o seu funcionamento baseia-se no uso de classes. Das classes individuais, a Graphics3D é provavelmente a mais importante, porque todo o *rendering* é feito através dessa classe. A classe World é crucial porque serve como raiz da estrutura do grafo de cena. A Object3D é a classe base de todos os objectos que podem ser desenhado ou carregados a partir de um ficheiro, e é também o lugar onde as animações são aplicadas. De seguida estas classes serão descritas mais detalhadamente.

Graphics3D

Esta classe é um contexto para os gráficos 3D que pode ser associado a um alvo de *rendering*. Todo o *rendering* é feito através dos métodos de desenho desta classe, incluindo o rendering dos objectos *World*. Não existe outra maneira de desenhar algo neste API.

World

Um versão especial do nó *Group* (usado pelo Java3D para criar grafos) que é o contentor de topo de grafos de cena.
Um grafo de cena é construído a partir de uma hierarquia de nós. Num grafo de cena completo, todos os nós estão invariavelmente ligados uns aos outros através de uma raiz comum, que é o nó *World*. A figura seguinte (Figura 9) é um exemplo de um grafo de cena completo.

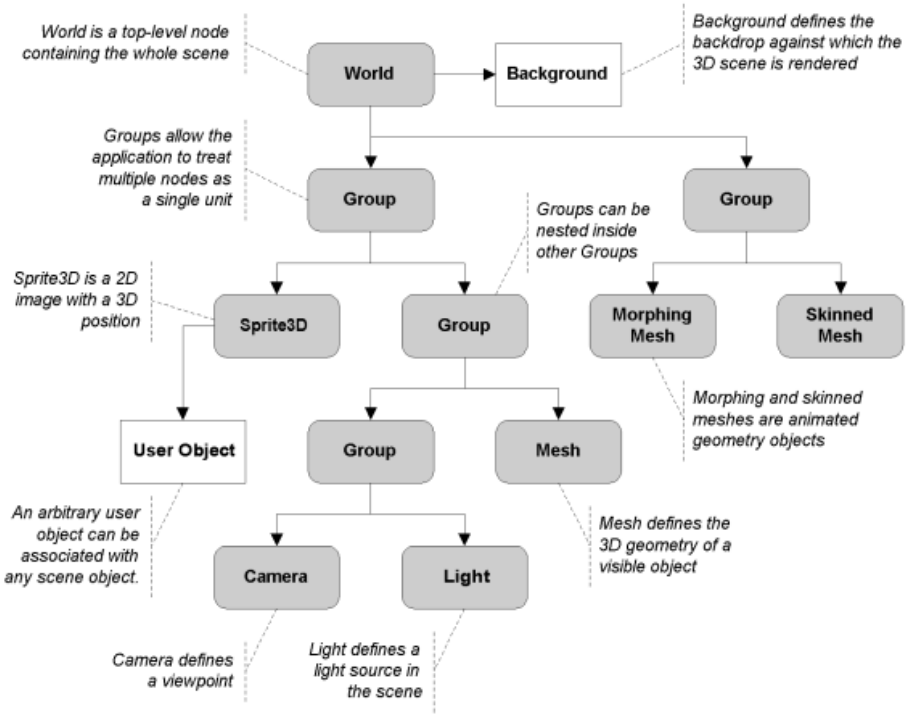

Figura 9 – Exemplo de grafo de cena, com o nó World no topo.

Refira-se que um grafo de cena não necessita de estar completo para ser desenhado, pois nós e ramos individuais podem ser desenhados usando um método separado na classe Graphics3D. Contudo a semântica para desenhar um grafo de cena incompleto é ligeiramente diferente quando comparado com o desenho de um *World*.

No entanto, apesar de ser chamado grafo, um grafo de cena é actualmente uma árvore. Isto implica que um nó só pode pertencer a um grupo de cada vez, e que o uso de ciclos é proibido. No entanto, objectos de componentes, como os *VertexArrays* (vectores de vértices), podem ser referenciados por um número arbitrário de nós e componentes.

Object3D

Esta é uma classe abstracta para todos os objectos que podem ser parte de um mundo 3D. Isto inclui o próprio *World*, nós de outros grafos de cenas, animações, texturas, etc. De facto, tudo neste API é um Object3D, excepto as classes Loader, Transform, RayIntersection e Graphics3D.

3.2.2.1 Exemplo de código

O exemplo de código seguinte mostra o uso de todas as classes fundamentais referidas anteriormente. Consiste na classe MyCanvas, que faz o carregamento remoto de um grafo de cena a partir de um ficheiro que se encontra num site http. Esse carregamento é feito para um vector de instâncias Object3D, sendo que o nó inicial é atribuído ao objecto myWorld que é uma instância da classe World. Depois, quando o método *paint* é invocado, é criado um objecto Graphics3D que vai ser responsável por desenhar o grafo de cena representado pelo myWorld.

```
public class MyCanvas extends Canvas
{
    World myWorld;
    int currentTime = 0;

    public MyCanvas() throws IOException {

        // Load an entire World. Proper exception handling is omitted
        // for clarity; see the class description of Loader for a more
        // elaborate example.

        Object3D[] objects =
Loader.load("http://www.example.com/myscene.m3g");
        myWorld = (World) objects[0];
    }

    // The paint method is called by MIDP after the application has issued
    // a repaint request. We draw a new frame on this Canvas by binding the
    // current Graphics object as the target, then rendering, and finally
    // releasing the target.

    protected void paint(Graphics g) {

        // Get the singleton Graphics3D instance that is associated
        // with this midlet.

        Graphics3D g3d = Graphics3D.getInstance();

        // Bind the 3D graphics context to the given MIDP Graphics
        // object. The viewport will cover the whole of this Canvas.

        g3d.bindTarget(g);

        // Apply animations, render the scene and release the Graphics.

        myWorld.animate(currentTime);
        g3d.render(myWorld);     // render a view from the active camera
        g3d.releaseTarget();     // flush the rendered image
        currentTime += 50;       // assume we can handle 20 frames per second
    }
}
```

3.2.3 Relação com o OpenGL ES

Um dos principais feitos do API foi terem conseguido conciliar as características base destes dois standards, M3G e OpenGL ES. Isto permitiu ao M3G ser implementado no cimo do OpenGL ES, em vez de haver dois motores autónomos num dispositivo apenas. Assim o M3G pode tomar partido de todas as vantagens do *hardware* gráfico sem ter que fazer alterações à especificação. No entanto, existe um motivo mais contundente para se ter optado pela utilização do OpenGL ES, ao invés de ser desenvolvido um motor próprio. O M3G é uma especificação bastante complexa, sendo apenas batida em termos de complexidade pelo MIDP no espaço J2ME. Seria impossível implementar tal complexidade em tão pouco espaço de tempo se não tivesse sido adoptado o OpenGL como base. A imagem seguinte (Figura 10) mostra como estão relacionados os diversos componentes de uma aplicação que use o M3G, e em especial a relação entre o M3G e o OpenGL ES.

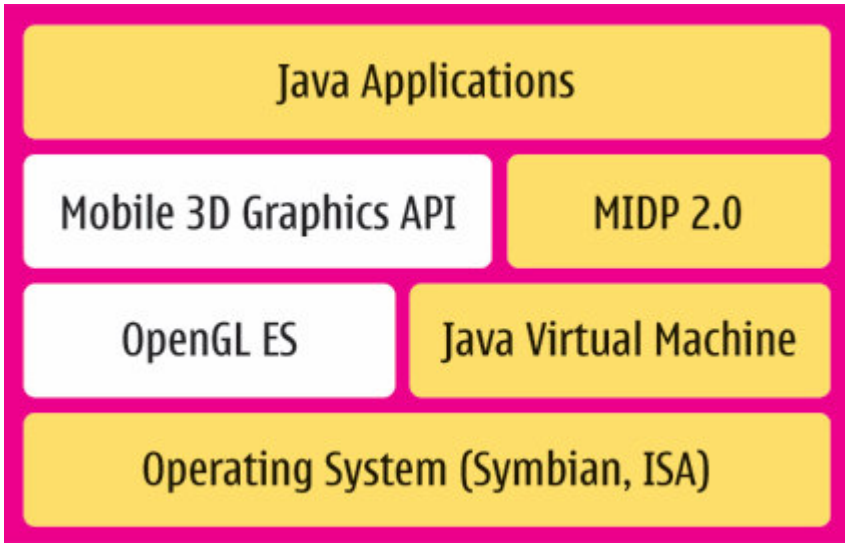

Figura 10 – Relação entre o M3G e o OpenGL ES

3.2.4 MIDP e CLDC

O MIDP (Mobile Information Device Profile) e o CLDC (Connected Limited Device Configuration) são ambientes que fazem parte da J2ME (Java 2 Platform, Mobile Edition). Este ambientes são fundamentais para o funcionamento do M3G.

O CLDC define um conjunto base de API's e uma máquina virtual para dispositivos limitados em recursos como telemóveis, pagers, e PDA's. O MIDP é usado em combinação com o CLDC e providencia um ambiente *runtime* base para os dispositivos referidos. O MIDP define uma plataforma para que se possa incorporar aplicações optimizadas, gráficas e/ou com acesso à rede de um modo seguro e dinâmico.

A biblioteca gráfica 3D encaixa em cima destes perfis, providenciando assim as suas funcionalidades gráficas avançadas.

3.2.5 Implementações

3.2.5.1 Superscape Swerve

A Superscape, companhia especializada em tecnologia 3D para Java e outros ambientes *wireless* nativos, foi a primeira a implementar o standard M3G (JSR-184) num produto. O pacote em que a empresa disponibiliza o M3G é designado por Swerve. Esse pacote vem em duas configurações possíveis, uma por *software* e outra por *hardware*. A versão por *software* não usa bibliotecas gráficas externas, enquanto a versão por *hardware* utiliza o OpenGL ES para executar todas as funções de *rendering* necessárias ao funcionamento do M3G, usando assim a aceleração gráfica presente no dispositivo.

3.2.5.2 Hybrid

A implementação M3G da Hybrid é construída no cimo da já analisada *framework* OpenGL ES da Hybrid. Esta implementação pode ser licenciada como um componente adicional à solução OpenGL ES existente.

Esta implementação não foge à regra, usando o OpenGL ES para a executar o código de processamento gráfico e acesso ao *hardware* gráfico 3D. No entanto, esta implementação é optimizada tendo em conta a máquina virtual Java presente no dispositivo, e se esta não for rápida o suficiente, essa parte do código é transferida de Java para C, de modo a manter uma boa performance. A Hybrid fornece a sua implementação OpenGL ES Gerbera se necessário, mas se já existir uma implementação OpenGL ES no dispositivo essa poderá ser utilizada.

3.2.6 Plataformas suportadas

As plataformas suportadas são todas aquelas que possuam uma implementação OpenGL ES e uma máquina virtual de Java presentes. No entanto, dependendo das implementações OpenGL ES e M3D presentes, poderão ter que ser feitos ajustes de modo a que fique tudo funcional.

3.3 JSR 239 – Java Bindings para OpenGL ES

O JSR (Java Specification Request) é um standard em desenvolvimento que tem como objectivo fazer ligações (*bindings*) OpenGL ES para o ambiente Java, mais propriamente, para J2ME (Java 2 Mobile Edition). O JSR 239 torna possível aceder ao API OpenGL ES directamente de uma aplicação Java. Esta especificação tenciona implementar dois perfis do OpenGL ES (já referidos anteriormente): Common e Common-Lite. O perfil Common-lite é apontado para os ambientes Connected Limited Device Configuration (CLDC) 1.0, enquanto o perfil Common é para o CLDC 1.1.

Esta especificação surge da necessidade de acesso a *hardware* acelerado a partir de uma biblioteca 3D de baixo nível, sendo que o OpenGL ES se está a tornar rapidamente o standard da indústria nessa área. É verdade que já existe um API 3D para dispositivos J2ME, já analisado anteriormente, o M3G (JSR 184). Este API providencia efectivamente a mesma funcionalidade que este JSR irá providenciar. No entanto, é orientado a objectos e desse modo a interface que permite aceder às funcionalidades é completamente diferente do OpenGL. Este JSR vai colmatar especialmente as necessidades dos programadores que já estão familiarizados com o OpenGL, e que não necessitam de nenhuma das funcionalidades de alto nível providenciadas pelo M3G. Adicionalmente, com a abundância de aplicações OpenGL que existe no mercado, uma API baseada na especificação do OpenGL irá facilitar os esforços de conversão dessas aplicações para a plataforma J2ME.

De referir ainda que para esta especificação funcionar assume-se a presença de uma biblioteca OpenGL ES no sistema.

3.4 Direct3D Mobile

O Direct3D Mobile da Microsoft é um API [6] que providencia suporte para gráficos 3D para as aplicações desenvolvidas para plataformas baseadas no Windows CE. É um derivado do API Direct3D encontrado nos sistemas *desktop* baseados no Windows da Microsoft, sendo optimizado para correr em plataformas portáteis.

O Direct3D Mobile compara-se mais aproximadamente ao Direct3D 8 do *desktop*, mas também implementa algumas elementos e comportamentos do Direct3D 9.

Os aspectos principais do Direct3D Mobile são os seguintes:

- Em comparação com o Direct3D para plataformas *desktop*, o tamanho do Direct3D Mobile é muito reduzido. Isto resulta da eliminação do suporte para alguns capacidade gráficas 3D que não estão disponíveis em plataformas móveis devido ao poder e *hardware* limitados.

- A arquitectura de drivers permite que se implemente soluções de *drivers* baseadas apenas em *software*, baseadas apenas em *hardware*, ou uma mistura de *software* e *hardware*.

- O Direct3D Mobile é construído com uma arquitectura multi-tipos. Em adição em aos valores com vírgulas flutuantes, o Direct3D Mobile suporta também valores de vírgula fixa no formato 16.16.

Para os programadores o Direct3D Mobile é um conjunto de interfaces COM. Esses interfaces são usados para especificar primitivas 2D e 3D, especificando como é que essas primitivas devem ser desenhadas, desenhando-as para um *framebuffer*, e finalmente mostrando esse *framebuffer* ao utilizador final.

3.4.1 Características

O Direct3D Mobile providencia acesso directo aos dispositivos de visualização enquanto mantém compatibilidade com o GDI (*graphics device interface*) do Windows, providenciando uma forma independente do dispositivo para as aplicações 3D e *software* de subsistema, como pacotes gráficos 3D, ganharem acesso a características específicas dos dispositivos de visualização.

Algumas das características avançadas do Direct3D são:

- Suporte para *z-buffers* 3D.
- *Depth buffering* alterável (usando *z-buffers* ou *w-buffers*).
- *Flat* e *Gouraud shading*.
- Múltiplas fontes e tipos de luz.
- Suporte total para texturas e materiais, incluindo *mipmapping*.
- *Drivers* de emulação por *software* precisas.
- Transformação e *clipping*.
- Independência de *hardware*.
- Suporte para a *Hardware Abstraction Layer* (HAL). Isto providencia uma interface consistente através da qual se trabalha directamente com o *hardware* de visualização, obtendo assim a performance máxima.
- Suporte para *page flipping* com múltiplos *back buffers* em aplicação de ecrã inteiro.
- Acesso a *hardware* de esticamento de imagem (*image-stretching*).
- Acesso ao *hardware* exclusivo.

3.4.2 Arquitectura

3.4.2.1 Processamento de Vértices e Pixeis

A parte do Direct3D que "empurra" geometria através do *pipeline* de geometria de função fixa é o motor de transformação. Ele localiza o modelo e o observador (*viewer*) no mundo, projecta vértices para serem mostrados no ecrã, e recorta os vértices para o *viewport*. O motor de transformação também executa computações de iluminação para determinar os componentes difusos e especulares em cada vértice.

O *pipeline* de geometria aceita vértices como *input*. O motor de transformação aplica três transformações – transformações do mundo, da vista (*view*) e projecção – aos

vértices, recorta o resultado, e passa tudo para o rasterizador. A imagem seguinte (Figura 11) mostra a sequência de passos.

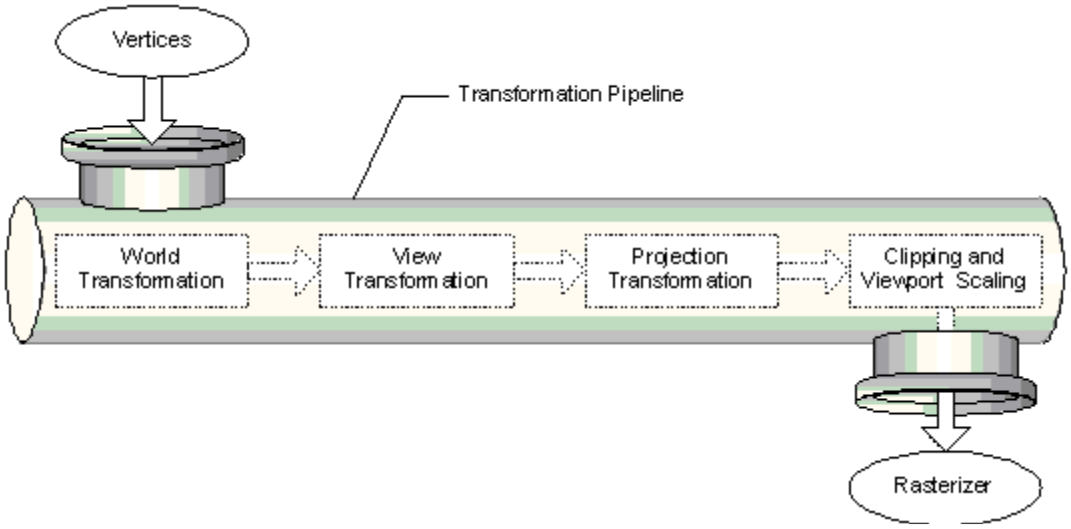

Figura 11 – *Pipeline* de geometria do Direct3D Mobile

No inicio do *pipeline* nenhuma das transformações foi aplicada, por isso todos os vértices do modelo são declarados relativos a um sistema de coordenadas local – isto é, um local de origem e uma orientação. Esta orientação de coordenadas é muitas vezes referida como espaço do modelo (*model space*), e as coordenadas individuais são chamadas coordenadas de modelo.

A primeira etapa do *pipeline* de geometria transforma os vértices de um modelo do seu sistema de coordenadas para um sistema de coordenadas que é usado por todos os objectos na cena. O processo de reorientar os vértices é chamado transformação do mundo (*world transformation*). Esta nova orientação é comumente referido como espaço de mundo (*world space*), e cada vértice no espaço de mundo é declarado usando coordenadas do mundo.

Na próxima etapa, os vértices que descrevem o mundo 3D estão orientados de acordo com a visão da câmara (*camera view*). Isto é, a aplicação escolhe um ponto de vista para a cena, e as coordenadas de espaço de mundo (*world space*) são recolocadas e rodadas em torno da visão da câmara, tornando o espaço de mundo em espaço de câmara (*camera space*). Esta é a transformação de vista (*view transformation*).

A próxima etapa é a transformação de projecção (*projection transformation*). Nesta parte do *pipeline*, os objectos são normalmente escalados de acordo com a sua distância do observador (*viewer*) de modo a dar a ilusão de profundidade a uma cena; os objectos mais próximos são feitos de modo a parecerem maiores que objectos distantes, e vice-versa. Este espaço que resulta da transformação de projecção é designado por espaço de projecção (*projection space*), embora alguns livros gráficos se refiram a ele como

espaço homogéneo após perspectiva (*post-perspective homogeneous space*). Nem todas as projecções de transformação escalam o tamanho dos objectos numa cena. Uma projecção dessas é designada por projecção ortogonal.

Na parte final do *pipeline*, qualquer vértice que não for visível no ecrã será removido, de modo a que o rasterizador não gaste tempo a calcular as cores e a iluminação de algo que nunca será visto. Este processo designa-se por *clipping*. Depois de ser feito o *clipping*, os vértices restantes serão escalados de acordo com os parâmetros do *viewport* e convertidos em coordenadas de ecrã. Os vértices resultantes – que são vistos no ecrã quando uma cena é rasterizada – existem no espaço de ecrã (*screen space*).

3.4.2.2 *Hardware* Abstration Layer (Camada de Abstracção de *Hardware*)

O Direct3D Mobile providencia independência de dispositivos com a *Hardware Abstration Layer* (HAL). A HAL é uma interface específica ao dispositivo, providenciado pelo fabricante do mesmo, que o Direct3D Mobile usa para trabalhar directamente com o *hardware* de visualização. As aplicações nunca interagem com a HAL. No entanto, com a infra-estrutura que a HAL fornece, o Direct3D Mobile dá acesso a um conjunto consistente de interfaces e métodos que uma aplicação usa para mostrar gráficos. O fabricante do dispositivo implementa a HAL em código de 32-bit, podendo ser parte de uma *driver* de *display* ou ser uma DLL (*dynamic-link library*) separada que comunica com a *driver* de *display* através de um interface privado que o criador da *driver* define.

A HAL apenas reporta as capacidades presentes no *hardware*, isto é, se uma determinada característica não é suportada pelo *hardware*, a HAL não faz qualquer emulação por *software* da característica. Ou seja, se uma função não é suportada pelo *hardware*, a HAL não a reporta como uma capacidade do *hardware*. Adicionalmente, a HAL não valida parâmetros, o Direct3D é que o faz antes de a HAL ser invocada.

3.4.2.3 Integração no sistema

A ilustração seguinte (Figura 12) mostra a relação entre o Direct3D Mobile, o *graphics device interface* (GDI), a *hardware abstraction layer* (HAL), e o *hardware*:

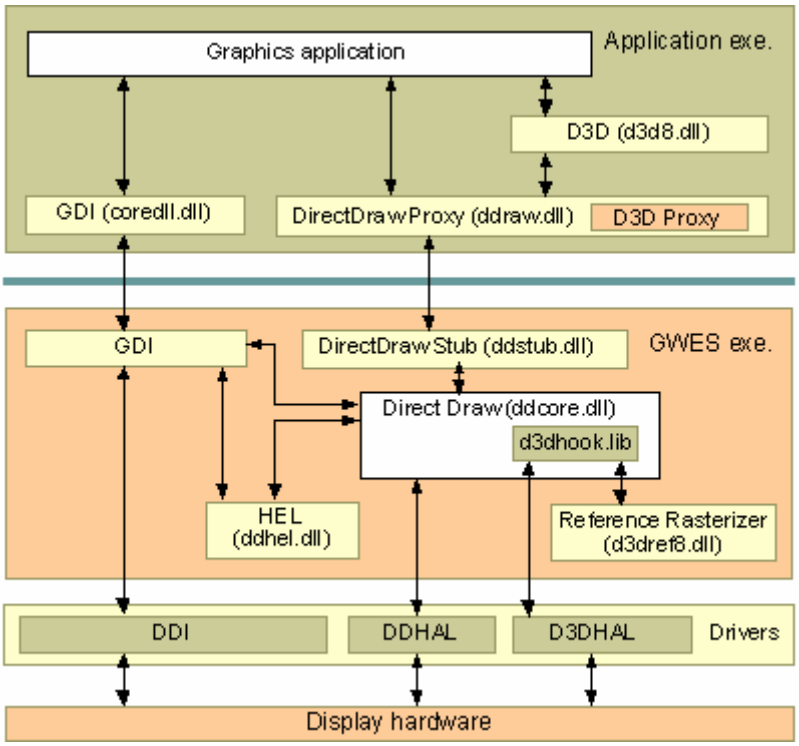

Figura 12 – Relação entre o D3D e o resto do sistema gráfico

Como o diagrama precedente mostra, as aplicações Direct3D Mobile existem lado a lado com as aplicações GDI e ambas têm acesso ao *hardware* gráfico através da *driver* do dispositivo. Ao contrário do GDI, o Direct3D Mobile pode tomar partido das características do *hardware* quando um dispositivo HAL é seleccionado. Os dispositivos HAL fornecem aceleração de *hardware* baseada no conjunto de características oferecido pelo chip gráfico do dispositivo. É disponibilizado pelo Direct3D Mobile um método para determinar em tempo de execução se um dispositivo é capaz de uma determinada tarefa.

3.4.3 Diferenças entre o Direct3D Mobile e o Direct3D 8 para sistemas *desktop* baseados no Windows

Como já foi referido anteriormente, o Direct3D Mobile é um subconjunto da funcionalidade providenciada pelo Direct3D 8 para sistemas *desktop*. A lista seguinte identifica as funcionalidades do Direct3D 8 que o Direct3D Mobile não suporta:

- *Bump mapping*
- *Cube maps*
- *Edge antialiasing*
- *Emissive materials* (materiais emissivos)
- *Gamma correction* (correcção Gamma)
- *Higher-order primitives*
- *Line stippling*
- *Mirror-once texture wrapping*
- *Multisample masking*
- *Non-local video memory*
- *Phong shading*
- *Pixel shaders*
- *Point sprites*
- *Spot lights*
- *State blocks* (blocos de estado)
- *Stereoscopic viewing*
- Mudar entre os modos processamento de vértices baseado em *hardware* e baseado em *software*
- *User clip planes*
- *Vertex blending* e *indexed vertex blending*
- *Vertex shaders*
- *Volume* ou *volume textures*

A lista seguinte identifica áreas de funcionalidade onde o Direct3D Mobile providencia suporte limitado:

- A inicialização do buffer de profundidade (*depth buffer*) é apenas suportada através de parâmetros de apresentação.

- O *Multisample antialiasing* é apenas suportado através de parâmetros de apresentação.

- Direct3D Mobile apenas suporta o dispositivo HAL, não suportando o *pure device* que é baseado exclusivamente em *hardware*.

- Apenas um dispositivo pode ser instanciado num momento e apenas o dispositivo por defeito é suportado.

- O Direct3D Mobile apenas suporta um único *vertex* e *index stream*.

3.4.4 Vantagens do Direct3D

- **Boa base de dispositivos baseados no Windows CE** no mercado, e com tendência a crescer.

- **Bom suporte por parte dos fabricantes de *hardware* gráfico** para plataformas móveis, como consequência da base de dispositivos existentes e do peso institucional da Microsoft.

- **Bastante documentação e profissionais com alguma experiência**, visto ser um subconjunto do Direct3D 8 para plataformas *desktop*, sendo que o API Direct3D é bastante popular e já existe há bastantes anos no mercado.

- **Baixo consumo de energia e uso de memória**, visto que o Direct3D Mobile foi especialmente desenhado para plataformas móveis, tendo assim em conta as suas limitações em termos de consumo de energia e memória.

- **Actualizações periódicas e retro-compatíveis ao API** de modo a reflectir a evolução do *hardware* gráfico são esperadas, tal como acontece na versão *desktop* do API.

3.5 Klimt

O Klimt é uma biblioteca 3D open-source, que tem como alvo telefones móveis e PDA's. Este API é muito semelhante aos do OpenGL e do OpenGL ES, o que é normal visto ser baseado no OpenGL ES. No entanto não reivindica conformidade com nenhum desses API's.

O Klimt tem como objectivos independência do *hardware* e velocidade. Os únicos requisitos que um dispositivo real necessita para ser capaz de correr o Klimt são um *framebuffer* linear de 16 bits (RGB565). O Klimt compila sem nenhuma dependência adicional. Não são necessárias bibliotecas adicionais para uma versão mínima.

3.5.1 Características

O Klimt actualmente providencia a seguinte funcionalidade:

- Suporte total para as matrizes de projecção e *modelview*
- Suporte total de matrizes de texturas
- Todos os tipos de primitivas definidos no OpenGL
- Vectores de vértices
- *Culling* e *clipping*
- Iluminação de vértices
- *Perspective* e *affine texturing*
- *Video background* (glDrawPixels)
- Listas de visualização (*Display lists*) e pilhas de atributos (*Attribute stacks*)

As seguintes características não são objectivos actuais do Klimt:

- *Polygon stippling*
- *Antialiasing*
- Acumulação e *alpha buffer*

- Geração automática de coordenadas de textura
- Todos os tipos de características que não são adequados a uma implementação rápida e simples

3.5.2 Comparação de características entre o Klimt, o OpenGL e o OpenGL ES

	OpenGL	OpenGL|ES	Klimt
glBegin/glEnd	✓	✗	✓
Tipos de Primitivas	Todos	Não existem Quads, Quad Strips e Polygons	Todos
Vectores de vértices (*Vertex Arrays*)	✓	✓	✓
Iluminação de vértices (*Vertex Lighting*)	✓	✓	✓
Tipos de dados (*Data Types*)	float, double, int, etc...	float, fixed	float, double, int, fixed
Multisampling e AA	Opcional	Opcional	✗
Polygon Stipple* e *Smooth	✓	✗	✗
glDraw/Read Pixels	✓	only glReadPixels	only glDrawPixels
Texturas	Todos os tipos	2D	2D
Texture wrap, repeat, etc…	✓	wrap, clamp_to_edge	wrap
Texturas comprimidas	✓	✓	✗
Multitexture	✓	Opcional	✗
Fog	✓	✓	✓
Scissor Test	✓	✓	✓
Alpha Test	✓	✓	✗
Stencil Test	✓	Opcional	✗
Depth Test	✓	✓	✓
Blending	✓	✓	✓
Clipplanes	✓	✗	✗
WGL Functions	✓	✗	✓
EGL Functions	✗	✓	✓

Tabela 4 – Comparação de características entre o Klimt, OpenGL e OpenGL ES

3.5.3 Arquitectura – Classes

As duas principais classes do Klimt são o klContext e o klRasterizer.

A classe klContext providencia a interface C++ do API (que é embrulhada por um C-wrapper) e usa a classe klRasterizer internamente. Todas as classes klEGL providenciam funcionalidade para acesso à memória vídeo. Um número de classes básicas (guardadas em src/base/) providenciam funcionalidades como vectores, matrizes e pilhas. A figura seguinte (Figura 13) mostra os três níveis de código presentes no Klimt. As classes apenas acedem a outras classes do mesmo nível ou em níveis inferiores.

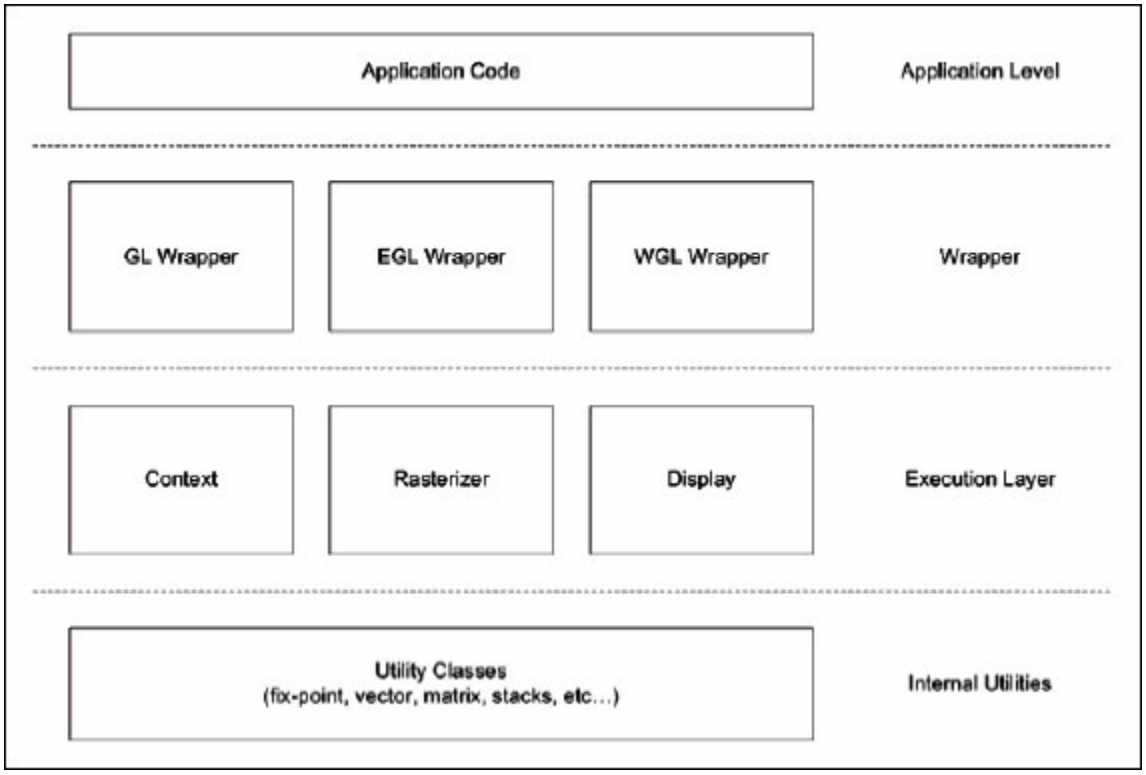

Figura 13 – Camadas de código do Klimt

3.5.4 Sistemas suportados

Actualmente o Klimt corre nas seguintes plataformas:

- Windows (XP, 2000, etc…)
- PocketPC 2002 e 2003
- MS Smartphone
- Linux no iPAQ (usando TinyPTC)
- Linux no Shart Zaurus
- Linux com X11

As plataformas seguintes são:

- SymbianOS (trabalho em andamento)
- PalmOS (planeado)

4 Comparação entre as bibliotecas 3D

Nas páginas seguintes serão analisadas e comparadas as bibliotecas 3D abordadas neste relatório. Essa comparação será feita utilizando várias perspectivas, podendo-se assim discernir os pontos fortes e fracos de cada uma das bibliotecas.

4.1 Sistemas suportados

Os sistemas portáteis actuais são inúmeros, existe milhares de combinações possíveis entre *hardware* (processadores, co-processadores, etc.) e sistemas operativos. Muitos desses sistemas portáteis não têm sequer capacidades para mostrar gráficos elaborados, tanto 2D como 3D (como é o caso alguns telemóveis e PDA's mais antigos), logo não são um alvo das bibliotecas 3D. Assim sendo, é muito difícil que qualquer uma das bibliotecas seja implementada em todos os sistemas, principalmente porque só agora é que os fabricantes começam a apostar no 3D para plataformas móveis.

É neste contexto que se pode afirmar que o OpenGL ES é a biblioteca 3D que abrange um maior número de sistemas, podendo-se mesmo dizer que a grande maioria das plataformas actuais têm implementações do OpenGL ES disponíveis, e alguns sistemas operativos vão mesmo incluir uma implementação do OpenGL ES nas suas próximas versões. Este é o caso do Symbian (na sua versão 8.0 [2]) e do PalmOS Cobalt [3]. Para garantir a compatibilidade com o sistema, existe a especificação EGL, que permite a interacção com o sistema. Isto é, para suportar uma nova plataforma pode ser implementado um EGL para o sistema em causa, não sendo necessário alterar o resto da implementação. Quanto às implementações analisadas, a da Hybrid leva clara vantagem, porque para além de já suportar um grande número de sistemas, possibilita a integração da sua plataforma OpenGL ES em vários outros sistemas por encomenda. Quanto ao Vincent, com as restrições normais em projectos *open source* (poucos colaboradores a tempo inteiro, falta de sistemas de teste), actualmente apenas suporta plataformas móveis baseadas no Windows Mobile e processadores da Intel baseados na arquitectura ARM.

O M3G também tem um bom suporte, embora nunca possa ser tão abrangente como o do OpenGL ES. A razão para isso é simples… o M3G precisa do OpenGL ES como suporte, logo para funcionar necessita que o sistema possua uma implementação OpenGL ES. Para além disso, como seria de esperar numa especificação Java, necessita de uma máquina virtual Java para correr. Logo, embora as máquinas virtuais de Java sejam muito populares nas plataformas portáteis, existem sempre sistemas que não as suportam. Assim sendo, o número de sistemas com implementações OpenGL ES e máquinas virtuais de Java será sempre inferior ao total de sistemas com implementações OpenGL ES.

Quanto ao futuro JSR 239, aplica-se exactamente o mesmo que foi dito em relação ao M3G, o que se justifica sabendo-se que ambos são standards Java.

O Direct3D Mobile é a única biblioteca 3D analisada que é totalmente proprietária, sendo pertença da Microsoft. Por isso não é de estranhar que apenas as plataformas baseadas no sistema operativo Windows CE sejam suportadas. Embora os sistemas baseados no Windows CE estejam longe de ser a maioria, não deixam de ser uma boa base de suporte. No entanto, sabendo-se que o Windows CE suporta máquinas virtuais de Java e já possui algumas implementações do OpenGL ES, o número de sistemas abrangido será sempre muito inferior ao das bibliotecas referidas anteriormente.

Por fim temos o Klimt. Embora seja *open source*, esta biblioteca já abrange um número razoável de sistemas operativos. Para além disso, fornecem algumas instruções de como fazer o *port* para novas plataformas, tarefa que pode ser executado por qualquer pessoa interessada. Assim sendo o número de plataformas possíveis é bastante elevado. No entanto tem a desvantagem de não ser um API apoiado pela indústria, pelo que não é de esperar que surjam implementações para plataformas mais "exóticas". No entanto, somando tudo, esta biblioteca tem uma abrangência bastante boa de sistemas, embora ainda lhe faltem implementações para dois sistemas operativos bastante populares, o Symbian e o PalmOS, estando no entanto já planeadas.

4.2 Características

As características suportadas actualmente são baseadas nas existentes nas bibliotecas 3D *desktop* que lhes serviram de suporte. Todas as características base são implementadas, sendo deixadas de fora as que não são adequadas às plataformas móveis, devido às limitações de processamento e de memórias existentes nas mesmas. O Direct3D Mobile é baseado no Direct3D 8 para *desktop*, embora aplique alguns comportamentos do Direct3D 9. As outras bibliotecas são todas directa ou indirectamente baseadas no OpenGL para *desktop*, nas suas versões 1.3 e 1.5.

É difícil dizer qual das bibliotecas implementa mais características. Por um lado, as bibliotecas baseadas em OpenGL possuem todas características bastante semelhantes. A única que não está directamente relacionada com a versão ES do OpenGL é o Klimt, embora em termos de características não fuja muito ao que existe na versão móvel do OpenGL. Quanto à questão da comparação entre as bibliotecas OpenGL e o Direct3D Mobile, é difícil dizer qual é a melhor neste caso. Nas plataformas *desktop* Windows ambas têm um suporte abrangente, e são lançadas novas versões e pacotes opcionais frequentemente para as bibliotecas estarem a par das novas tecnologias. Sendo as variantes móveis reflexos directos das versões *desktop*, podemos assumir que em termos de características, à falta de informação pormenorizada, estarão todas as bibliotecas ao mesmo nível.

Existe no entanto uma característica que é fundamental implementar nas plataformas móveis, e que nem sequer necessita de ser equacionada pelas versões *desktop* das bibliotecas 3D. Essa característica é o suporte para vírgula fixa. Nos sistemas *desktop* a existência de uma unidade de vírgula flutuante é dada como adquirida, pois todos os processadores modernos possuem uma ou mais dessas unidades. No entanto, nas

plataformas móveis são escassos os sistemas que possuam suporte para vírgula flutuante, tornando-se fundamental uma alternativa. Essa alternativa é o uso de vírgula fixa. Todas as bibliotecas 3D e respectivas implementações suportam vírgula fixa, no entanto nem todas possuem suporte para vírgulas flutuantes.

O OpenGL ES especifica dois perfis: o perfil Common, que suporta tanto vírgula fixa como vírgula flutuante; e o perfil Common-Lite, que apenas suporta vírgula flutuante. A implementação Gerbera da Hybrid suporta o perfil Common, ou seja, tanto providencia vírgula fixa como flutuante. A implementação Vincent apenas suporta o perfil Common-Lite, não fornecendo portanto suporte para vírgula flutuante. Quanto às especificações Java, o suporte ou não da vírgula flutuante depende da implementação OpenGL ES presente no sistema. Já o Klimt e o Direct3D Mobile suportam tanto a vírgula fixa como a flutuante. De referir ainda que algumas bibliotecas suportam um tipo optimizado de vírgula fixa para processadores XScale da Intel usando o Windows CE, implementado através da livraria Graphics Performance Primitives da Intel (Intel GPP). Esta livraria é utilizada no Klimt e no Vincent, desconhecendo-se se é implementado nas outras bibliotecas.

4.3 Arquitectura

Em termos de arquitectura existe uma predominância de API's baseadas no OpenGL, que como tal herdam o *pipeline* de transformação do mesmo (Figura 14). No entanto, os restantes aspectos da arquitectura variam entre várias bibliotecas, destacando-se nesse aspecto o M3G. Por outro lado temos o Direct3D Mobile da Microsoft, que tem uma abordagem própria e que se distancia das restantes, até por não estar relacionada com o OpenGL.

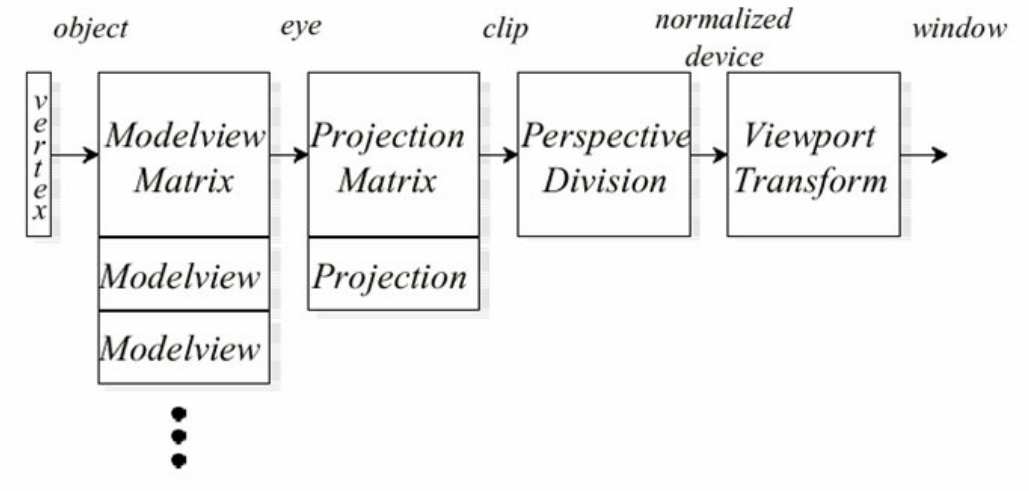

Figura 14 – *Pipeline* de transformações do OpenGL

A especificação OpenGL ES baseia-se completamente na arquitectura na sua versão *desktop* (OpenGL 1.3 e 1.5), usando assim os mesmos *pipelines*. No entanto, as implementações têm abordagens diferentes para conseguir esse objectivo. O Gerbera da Hybrid tem uma estrutura baseada em componentes de código, um para cada função. Com esta arquitectura, a implementação ganha flexibilidade, pois para alterar ou melhorar alguma das partes basta substituir o componente dessa parte, mantendo-se o resto do código inalterado. Por exemplo, para alterar a estrutura para suportar vírgula flutuante ao invés de vírgula fixa, basta substituir a livraria matemática por uma que suporte uma unidade de vírgula flutuante. Quanto ao Vincent, não se encontra dividido em componentes, sendo fornecida uma série de funções correspondentes que mapeia a API em chamadas da subjacente implementação C++. De notar que ambas as implementações utilizam implementações EGL para interagir com os sistemas que suportam.

O Klimt usa camadas de código para executar as várias funções necessárias para criar as imagens e objectos 3D. Essas camadas são quatro. Uma só para implementar utilidades que são fundamentais para o resto da biblioteca, como classes para vírgula fixa, vectores, matrizes e pilhas. A um nível superior existe uma camada de execução, que contem o rasterizador, classes para contexto e para visualização. No nível seguinte existe uma camada de encapsulamento, que contém várias classes de encapsulamento para várias plataformas, incluindo classes EGL e WGL. Por fim temos a camada de aplicação, que é onde está o código desenvolvido para criar aplicações.

O M3G, embora tenha o OpenGL ES como base, opta por uma abordagem completamente diferente. A abordagem é de mais alto nível, sendo orientada a objectos e usando um grafo de cena para definir o mundo. Assim sendo, todas as primitivas são definidas através de classes e tudo se traduz em objectos. Existe uma classe que é responsável por tudo o que se desenha. Existe outra classe de onde deriva praticamente tudo o que pode ser desenhado no ecrã. Por último, há ainda outra classe que é usada para aglomerar todos os objectos do mundo, no grafo já referido. Existe ainda classes para fazer várias transformações, carregar objectos, entre outras coisas. É de facto uma arquitectura completamente diferente da usada tanto pelos derivados pelo OpenGL e pelo Direct3D Mobile, mas no entanto é apenas um API de alto nível, visto que tudo é realmente posto no ecrã usando uma implementação OpenGL ES que tem obrigatoriamente de estar presente no sistema.

Por último, falta falar do Direct3D Mobile. Este API usa, tal como o OpenGL, um *pipeline* geométrico para fazer as transformações necessárias aos objectos de modo a que possam ser mostrados no ecrã (Figura 15). Para interagir com o *hardware* gráfico, o Direct3D Mobile precisa de uma *Hardware Abstraction Layer* (HAL), que normalmente é implementada no driver desse *hardware*. O Direct3D Mobile nunca interage directamente com o *hardware*, mas sim com o HAL. No entanto, o Direct3D Mobile não necessita da presença obrigatória de uma HAL para efectuar o *rendering*, usando funções por *software* para o efeito em caso de falta de *hardware* específico. O mesmo acontece quando o *hardware* gráfico não suporta todas as funcionalidades do Direct3D Mobile. O API consegue determinar em tempo real, usando a HAL, se uma determinada função é ou

não suportada pelo *hardware*, e caso não seja utiliza a implementação por *software* dessa funcionalidade.

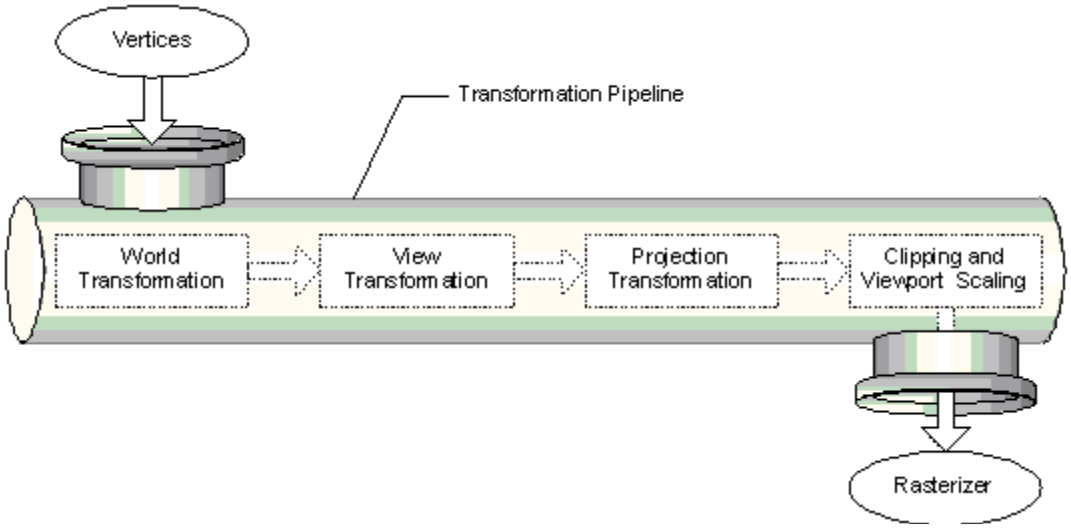

Figura 15 – *Pipeline* de transformações do Direct3D

4.4 Aceleração gráfica

Neste capítulo, quase todas as bibliotecas estão em pé de igualdade. A excepção é claramente a biblioteca *open source* klimt.

O OpenGL ES trata-se de um standard da indústria, que contempla a aceleração por *hardware* na sua especificação. Como tal, todo o *hardware* gráfico 3D para plataformas móveis suporta este API. Como consequência, todas as bibliotecas que usam o OpenGL ES como base (M3G, JSR 239) também têm disponível a aceleração gráfica.

O Direct3D Mobile é um API também suportado pelos fabricantes de *hardware* gráfico, existindo *Hardware Abstraction Layers* (HAL) nos drivers para todo o *hardware* gráfico analisado.

Quem fica a perder é o Klimt, que não procura utilizar *hardware* gráfico, até pelo facto de não ter grande suporte por parte da indústria.

A tabela seguinte (Tabela 5) mostra as séries de *hardware* gráfico referidas anteriormente e o seu suporte relativo às bibliotecas 3D analisadas.

Hardware Biblioteca	ATI Imageon 23xx	Bitboys Gxx	Imagination PowerVR MBX	Nvidia GoForce 3D
OpenGL ES	✓	✓	✓	✓
M3G	✓	✓	✓	✓
JSR 239	✓	✓	✓	✓
Direct3D Mobile	✓	✓	✓	✓
Klimt	✗	✗	✗	✗

Tabela 5 – Suporte às bibliotecas 3D por parte de vários fabricantes de *hardware* gráfico

4.5 Performance

É difícil medir qual das bibliotecas tem maior performance, no entanto pode-se inferir quais delas terão menor performance. Quanto à diferença de performance entre o Direct3D Mobile e o OpenGL ES, não é fácil comparar, até porque são API's com bases completamente diferentes e não haver ferramentas de medição de performance. Para além disso, é difícil dizer qual das bibliotecas é melhor devido à existência de vários sistemas operativos e *hardware* gráfico, que tornam ainda mais complicado dizer qual é a biblioteca mais rápida globalmente. No entanto, entre as bibliotecas baseadas no OpenGL é possível deduzir quais delas terão mais performance.

O OpenGL ES é de facto a biblioteca com condições para ser mais rápida. Das suas implementações, a Gerbera da Hybrid leva vantagem pois optimiza as suas implementações tendo em conta o sistema operativo e os processadores, enquanto a implementação Vincent está apenas optimizada para Windows CE e processadores XScale. O M3G é uma biblioteca 3D de alto nível e orientada a objectos, usando o OpenGL ES como base. Como tal, a sua performance é penalizada e nunca será tão boa como a da implementação OpenGL ES que lhe serve como base. Quanto ao futuro JSR 239, sofre do facto de ser uma camada Java para aceder ao OpenGL ES, nunca poderá ser tão rápido como a implementação OpenGL ES que utiliza. No entanto, o JSR 239 deverá provar ser mais rápido que o M3G, pois consiste apenas em *bindings*, não tendo uma elaborada estrutura orientada a objectos. No patamar mais baixo de performance está, pelo menos teoricamente, o Klimt. O facto de ser a única que não tem suporte para aceleração gráfica é uma grande desvantagem, não podendo assim aproximar-se da performance das outras bibliotecas.

4.6 Vista Geral

Neste momento existem três bibliotecas gráficas 3D que se destacam, já possuem algum suporte da indústria, e se preparam para assumir uma posição dominante na mesma indústria. São elas o OpenGL ES, o M3G e o Direct3D. Quanto ao OpenGL ES, várias versões de sistemas operativos para plataformas portáteis já vêm com

implementações destas API's incluídas, e a tendência é para aumentar. O facto de ser a versão portátil do API OpenGL dá-lhe um estatuto difícil de igualar por qualquer outra biblioteca 3D. Quanto ao M3G, é a primeira abordagem do Java aos gráficos 3D nas plataformas 3D. Visto que o Java é bastante popular nas plataformas móveis, esta biblioteca tem sido bastante utilizada, até pelo apoio que tem tido por parte da Nokia, que inclusive foi a impulsionadora do Java Specification Request (JSR 189) que lhe deu origem. Por fim, o Direct3D Mobile tem o peso de ser uma biblioteca criada pela Microsoft, o que faz com que esteja implementado de raiz em todos os sistemas operativos para plataformas portáteis baseados no Windows CE, que são bastante populares em PDA's (PocketPC's) e Smartphones.

Restam o JSR 239 e o Klimt. O JSR 239 tem algum potencial, visto que aplica o uso quase directo do OpenGL ES ao Java. No entanto pode ter a sua utilidade limitada, visto que se pode simplesmente usar o OpenGL ES directamente usando uma linguagem tipo C, e sem a perda de performance associada. Quanto ao Klimt, o facto de ser um projecto *open source* com pouco peso na indústria, e o facto de não providenciar aceleração por *hardware*, limitam um bocado a potencialidade desta biblioteca.

5 *Middleware*

Existem vários tipos de *software* que facilitam a tarefa do programador ao adicionar uma camada entre as ferramentas (Bibliotecas 3D, API's dos sistemas operativos, etc.) e o programador, de modo a lhe simplificar vida, simplificando processos e adicionando funcionalidades de mais alto nível. Esse tipo de *software* é designado por *middleware*, e também existe nas plataformas móveis relacionado com bibliotecas 3D.

Estes são alguns dos pacotes de *middleware* relacionados com gráficos 3D mais populares para plataformas móveis:

- **Fathammer X-Force 2** – O X-Force 2 é um pacote completo de desenvolvimento de jogos que tem um suporte avançado para OpenGL ES, usando para isso implementações da Hybrid. Este *middleware* providencia suporte para vários conjuntos de sistema operativo/plataforma e inclui optimizações para as arquitecturas de processadores mais comuns, como a Intel XScale e a TI OMAP.

- **Criterion Renderware** – O Renderware é um kit de desenvolvimento de jogos multi-plataforma, que tem uma versão especializada em plataformas móveis. Nessa versão utiliza a *framework* da Hybrid para providenciar implementações OpenGL ES e M3G. Tem parcerias com a Intel, Renesa e TI, providenciando assim suporte para os processadores XScale, SH Mobile e OMAP.

- **Synergenix mophun** – O mophun é mais kit de desenvolvimento de jogos para plataformas portáteis. Um dos seus componentes designa-se por mophun 3D, e tem como base uma implementação OpenGL ES desenvolvida pela própria Synergenix. Este componente providencia assim gráficos 3D nas plataformas suportadas.

Como se pode ver pela amostra, a maior parte do *middleware* que contém acesso a bibliotecas 3D está relacionado com o desenvolvimento de jogos, pelo que se prevê que este seja um dos usos por excelência das mesmas bibliotecas.

6 O futuro

A evolução das plataformas móveis deverá atenuar algumas das limitações existentes actualmente. Deverão surgir novas baterias mais duradouras, processadores cada vez mais rápido sem que haja grande aumento no consumo de energia, mais memória e ecrãs cada vez maiores e com mais resolução e profundidade de cor. A par destas inovações começará a surgir tecnologia gráfica, tanto embutida nos processadores e co-processadores, como em co-processadores próprios já com alguma memória associada. Essa tecnologia suportará as bibliotecas 3D mais populares e teremos gráficos 3D cada vez mais complexos e elaborados em dispositivos que cabem na palma das nossas mãos.

A tecnologia gráfica virá principalmente de duas formas: nos SoC (System on Chip) e nos co-processadores gráficos. O conceito de SoC consiste no fabrico de processadores que chamam a si todas as funcionalidades necessárias para o funcionamento do sistema, desde o processamento em si à aceleração gráfica, passando pelo controlo de energia e interfaces usb e bluetooth. Um exemplo desse conceito é o OMAP2420 da Texas Instruments [4], que usa a propriedade intelectual PowerVR MBX para a aceleração 2D/3D. Os co-processadores são independentes do processador, tendo por vezes memória própria, devendo ter mais performance pois são uma solução 100% dedicada.

Como tal, com o evoluir da tecnologia gráfica, as principais bibliotecas 3D desenvolver-se-ão e teremos uma dinâmica semelhante à existente nos *desktop*, em que surgem novas versões das bibliotecas gráficas com alguma regularidade, acompanhando e puxando pela tecnologia.

7 Conclusão

As bibliotecas gráficas 3D para plataformas móveis estão a dar os primeiros passos. As especificações estão nas primeiras versões, ainda não é fácil encontrar implementações (principalmente gratuitas). No entanto é fácil perceber que a indústria se encontra receptiva. Os principais fabricantes de *hardware* (gráfico e não só) estão a movimentar-se de modo a oferecer soluções que permitam aceleração 2D e 3D compatível com as principais bibliotecas gráficas existentes. E mesmo não havendo ainda plataformas com capacidades gráficas 3D avançadas no mercado, existe já um bom suporte por parte de ferramentas de *software* para o desenvolvimento de aplicações 3D.

Podemos então depreender que pelo futuro das plataformas móveis passam pelos gráficos 3D, e como tal, as bibliotecas 3D estão para ficar. No entanto, apenas as bibliotecas mais populares deverão sobreviver ao teste do tempo e continuarão a lançar novas versões para acompanhar a tecnologia. As principais candidatas a esse papel já se encontram definidas (OpenGL ES, M3G e Direct3D Mobile), restando saber se poderá existir alguma surpresa, surgindo e/ou impondo outra biblioteca.

8 Referências

[1] http://www.futuremark.com/pressroom/pressreleases/?070704

[2] http://www.mobilemag.com/content/100/103/C2519/

[3] http://www.palmsource.com/press/2004/032204_khronos.html

[4] http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11990&path=templatedata/cm/product/data/omap_2420

[5] http://sourceforge.net/docman/display_doc.php?docid=23735&group_id=87506

[6] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemultimedia5/html/wce50oriDirect3DMobile.asp

[7] http://www.khronos.org/

[8] http://www.khronos.org/opengles/index.html

9 Bibliografia

Bibliotecas 3D:

Site oficial da especificação OpenGL ES:
- http://www.khronos.org/opengles/

Site oficial da Hybrid e das suas implementações OpenGL ES e M3G:
- http://www.hybrid.fi/

Site oficial da implementação OpenGL ES Vincent:
- http://ogl-es.sourceforge.net/

Site da Qualcomm com informação útil sobre o OpenGL ES:
- http://www.cdmatech.com/solutions/products/q3d_why3d.jsp

Página do fórum Nokia relativa ao M3G (JSR 184):
- http://www.forum.nokia.com/main/1,,1_0_10,00.html#jsr184

Artigo da Nokia referente ao M3G (JSR 184):
- http://www.nokia.com/nokia/0,,62395,00.html

Site do Java Specification Request 184:
- http://www.jcp.org/en/jsr/detail?id=184

Site do Java Specification Request 239:
- http://www.jcp.org/en/jsr/detail?id=239

Site do MSDN relativo ao Direct3D Mobile:
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemultimedia5/html/wce50oriDirect3DMobile.asp

Site oficial da biblioteca 3D Klimt:
- http://studierstube.org/klimt/

***Hardware* Gráfico:**

Site da empresa Bitboys e da sua série de processadores gráficos Gxx:
- http://www.bitboys.com/

Site oficial da linha Imageon 23xx da ATI:
- http://www.ati.com/products/imageon2300/

Site do SDK da Nvidia relativo ao seu processador gráfico GoForce3D:
- http://developer.nvidia.com/object/hhsdk_home.html

Site oficial núcleo gráfico PowerVR MBR:
- http://www.powervr.com/Products/Graphics/MBX/Index.asp

***Middleware*:**

Site oficial do sistema X-Forge da Fathammer:
- http://www.fathammer.com/id/technology/index.html

Site oficial da plataforma Renderware da Criterion:
- http://www.fathammer.com/id/technology/index.html

Site do sistema mophun da Synergenix:
- http://www.mophun.com/

Sistemas Operativos:

Página do sistema operativo Windows Mobile e derivados:
- http://www.microsoft.com/windowsmobile/default.mspx

Página do sistema operativo PalmOS:
- http://www.palmos.com/

Página do sistema operativo Symbian:
- http://www.symbian.com/

Processadores:

Lista de processadores ARM:
- http://www.geocities.com/~banksp/Archives/ARMChips.html

Lista de fabricantes de processadores ARM:
- http://www.bluewatersys.com/consulting/doc/resources/chipvendors.php

Documento com o guia de design para PC98:
- http://tech-www.informatik.uni-hamburg.de/lehre/ss2002/pc-technologie/docs/pc98.pdf

Documento com a guia de design para PC99:
- http://www.colombus.cu/DrTaco/hard/docs/PC99_Design_Guide.pdf

10 Glossário

API (*application programming interface*) – um API consiste numa série de funções que permitem aceder a funcionalidades de baixo nível, como funcionalidades de hardware ou de sistema operativo.

Benchmark – um benchmark é consiste num conjunto de testes usado para testar a performance e/ou usabilidade de uma peça hardware ou software.

Binding – é o acto de ligar alguma coisa a outro. No caso da informática, consiste, por exemplo, em implementar numa linguagem o acesso a uma função de outra linguagem.

Buffer – é uma área de memória usada temporariamente pelos programas de modo a que a sua execução seja mais rápida ou eficiente.

Desktop – é o vulgar PC de secretária, usado tanto para fins pessoais como para estações de trabalho.

DLL (*dynamic link library*) – uma DLL é uma livraria usada pelas várias plataformas Windows da Microsoft, cujas funcionalidades podem ser partilhadas por uma multiplicidade de aplicações.

DSP (*Digital Signal Processor*) – é um processador ou co-processador especializado em processar dados digitais, que podem ir desde som até fluxos de dados.

Framebuffer – é um buffer especializado que é utilizado pelas bibliotecas gráficas para guardar as várias imagens que vão sendo processadas. A sua implementação pode ser feita de várias maneiras diferentes.

Framework – é uma estrutura lógica onde os vários elementos de uma arquitectura se encontram interligados. No caso da informática, uma framework pode ser um conjunto de ferramentas que se encontram ligadas entre si.

Hardware – em termos informático é qualquer componente físico relacionado com um computador.

IEEE – é uma organização ligada à electrotecnia que é responsável pela criação de vários standards reconhecidos mundialmente.

Mipmap – é uma técnica usada para melhorar a qualidade de imagem de objectos 3D distantes. Tal é conseguido implementando diferentes qualidades de texturas conforme a distância a que se encontra o objecto, de modo a dar uma sensação de profundidade e a não haver distorção de texturas.

Open-Source – consiste num programa de computador cujo código fonte está disponível livremente.

Pipeline – é uma espécie de "tubo" onde por um lado entram dados não tratados (vértices por exemplo) e, depois de várias transformações ao longo do "tubo", surgem dados tratados.

Rendering – consiste na conversão de uma descrição baseada em objectos de alto nível (como a que é feita nas bibliotecas 3D) numa imagem gráfica para ser visualizada.

SDK (Software Development Kit) – é um conjunto de ferramentas que facilitam ao programador o desenvolvimento de software para uma dada plataforma.

Software – é o termo genérico usado para todos os tipos de programas usado para operar computadores e dispositivos relacionados.

Sprite – é uma imagem animada individualmente numa imagem maior ou num conjunto de imagens.