3. Interfaces de Programação de Aplicativos (APIs)

O termo API significa *Application Programming Interface* (ou Interface de Programação de Aplicativos). A API é uma das ligações entre o *hardware* e o *software* (a outra ligação são os *drivers*). A API interpreta e comunica o *hardware* com o *software*. Deste modo, o programador não constrói programas para um *hardware* especificamente, ele pode o fazer de duas formas: para o sistema operacional, onde um *driver* traduzirá o que o programa requer e acessará o *hardware* de acordo com o que foi pedido. Ainda, o programa pode ser construído para uma interface de programação com a mesma finalidade. Através desta técnica não é necessário que o programador compreenda as placas existentes, este papel quem fará é o *driver* convertendo o que o programador quer em um comando compatível.

Atualmente, o OpenGL e o DirectX são as APIs mais utilizadas para gráficos (jogos, por exemplo). Um jogo é escrito utilizando comandos do OpenGL ou DirectX e não para uma placa específica. A API escolhida converterá os comandos enviados pelo jogo em instruções que os *hardware* entendam - como placa de vídeo e placa de som. Do mesmo modo que um programa, existem várias versões das APIs. Quando um jogo é da versão *X* do OpenGL ou do DirectX (aqui descrito de forma genérica, em um programa normalmente usa-se uma das duas APIs somente), significa que este jogo usa instruções do OpenGL *X* ou DirectX *X* (DirectX 9.0c, por exemplo). Para que este jogo funcione adequadamente, é necessário ter instalado a API na versão certa ou versão superior e preferencialmente um *hardware* também da mesma geração ou superior. Se, por exemplo, um jogo for DirectX *X* e o computador que irá rodá-lo tiver placa de vídeo com chip gráfico de versão inferior, quando o jogo requisitar um comando que a placa de vídeo não poderá executar, o DirectX fará uma emulação para executar o comando e o jogo não travar. A emulação não é perfeita, por isto a qualidade da imagem do jogo pode ficar afetada.

OpenGL vs DirectX

OpenGL e DirectX são consideradas as duas melhores APIs para processamento 2D e 3D. Ambas utilizam para a construção de seus mundos virtuais ou gráficos, algoritmos e teoremas da área da computação gráfica. Estas APIs têm implementados algoritmos importantes, o que significa que o programador não precisa desenvolvê-los, *reinventando a roda*. Enfim, qual é considerada a melhor API? O DirectX possui uma estrutura completa para manipular som, vídeo, rede, entre outras. O OpenGL é somente uma API gráfica. Por outro lado, o OpenGL é uma API *open source*. Além disto, provê suporte para vários sistemas operacionais, enquanto que o DirectX funciona somente no Windows e no X-Box (video-game da Microsoft).

Com as duas APIs é possível desenhar pontos, linhas, triângulos, quadrados, entre outras formas. Ainda, estas APIs dão suporte à aplicação de texturas. Ambas guardam informações dos objetos no espaço virtual e a partir da posição da câmera, renderizam os mesmos na tela. Como a maioria das aplicações são dinâmicas e realizam mudanças na câmera que devem ser repassadas de forma ágil ao

usuário, as APIs utilizam a capacidade de *hardware* para realizar processamentos dos vértices, faces, etc. Conseqüentemente, quem queira trabalhar com objetos 3D renderizados em tempo real, precisa de uma potente placa de vídeo aceleradora 3D.

Outra diferença é que a API DirectX foi feita em um modelo orientado a objetos, enquanto que no OpenGL, o programador terá que escolher dentre inúmeras funções. Pode ser que esta seja uma vantagem, para um desenvolvedor que irá criar sua própria *engine* (gerenciador temporal, físico e organizacional do mundo em 3D) o fato de ter a possibilidade de controlar como os objetos são renderizados pode resultar em ganho de performance ou qualidade visual.

Nas subseções seguintes estão descritas as APIs OpenGL e DirectX.

3.1 OpenGL

Na década de 80, cada fabricante de *hardware* tinha seu próprio conjunto de instruções para desenho de gráficos 2D e 3D. Construir aplicações com essas tecnologias que suportassem vários *hardware* era um verdadeiro desafio. Este esforço era enorme e havia pouco espaço para companhias menores e sem capital para tamanho investimento. Em meados dos anos 80, um padrão surgiu na indústria *Programmer's Hierarchical, Graphics System* – PHIGS, entretanto era complexo e desatualizado.

No final desta década foi criado um padrão chamado Iris GL. Este padrão era mais fácil de usar, além de ser considerado o estado da arte de uma API gráfica. Assim, esta API tornou-se pública, possibilitando todos os fabricantes de *hardware* a adotá-la. Em conseqüência às limitações da API Iris surgiu o padrão OpenGL. Desde 1992, o padrão é mantido pelo *Architecture Review Board* – ARB, um conselho formado por empresas como a 3DLabs, ATI, Dell, Evans&Sutherland, HP, IBM, Intel, NVIDIA, Sun e Silicon Graphics. A função desse conselho é manter a especificação e indicar quais recursos serão adicionados a cada versão.

No entanto, os projetistas do OpenGL tinham dois desafios. O primeiro é que os fabricantes de *hardware* queriam adicionar recursos próprios, sem que tivessem que esperar para esses recursos estarem oficialmente aceitos no padrão. Para resolver esse problema, incluíram uma maneira de estender o OpenGL. Muitas vezes, essas extensões são interessantes o suficiente para que outros vendedores de *hardware* as adotem. Então, quando elas são consideradas suficientemente abrangentes ou suficientemente importantes, a ARB pode promovê-las para que façam parte do padrão oficial. Praticamente todas as modificações recentes do padrão começaram como extensões, a grande maioria devido ao mercado de jogos.

Em segundo lugar, os projetistas também sabiam que muitos *hardware* não seriam poderosos o suficiente para abranger todo o padrão. Por isso, também incluíram extensões de *software*, que permitiam emular as funcionalidades necessárias, ou seja, mesmo com um *hardware* não muito avançado é possível empregar o padrão utilizando também as extensões que suprem estas deficiências.

Assim, seria resolvido um problema sério que ocorria com o antigo padrão Iris que era o da aplicação simplesmente não rodar pela falta de um ou outro recurso.

Arquitetura

O OpenGL é uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), portável e rápida. OpenGL não é uma linguagem de programação, é uma API para criação de aplicações gráficas. Seu funcionamento é semelhante ao de uma biblioteca C, uma vez que fornece uma série de funcionalidades. Quando é dito que um programa é baseado em OpenGL ou é uma aplicação OpenGL, significa que é escrito em alguma linguagem de programação que faz chamadas a uma ou mais bibliotecas OpenGL. As aplicações desta biblioteca podem ser desde ferramentas CAD (*Computer Aided Design*) a programas de modelagem usados para criar personagens para o cinema.

O OpenGL é um conjunto de comandos e funções que fornecem acesso a praticamente todos os recursos do *hardware* de vídeo. Esta API inclui aproximadamente 250 comandos e funções (200 comandos do *core* OpenGL e 50 da GLU – *OpenGL Utility Library*). Por ser portável, o OpenGL não possui funções para gerenciamento de janelas, interação com o usuário ou arquivos de entrada/saída. Cada ambiente (exemplo o *Microsoft Windows*) possui suas próprias funções para estes propósitos. O OpenGL fornece um pequeno conjunto de primitivas gráficas para construção de modelos, tais como pontos, linhas e polígonos.

Internamente, o OpenGL age como uma máquina de estados, que de maneira bem específica dizem ao OpenGL o que fazer. Usando as funções da API, podem ser ativados ou desativados vários aspectos dessa máquina, tais como a cor atual, se transparência será usada, se cálculos de iluminação devem ser feitos, se haverá ou não o efeito de neblina, assim por diante. É importante conhecer cada um desses estados, pois não é incomum a obtenção de resultados indesejados simplesmente por deixar um ou outro estado definido de maneira incorreta.

O OpenGL foi projetado para funcionar mesmo que a máquina que esteja exibindo os gráficos não seja a mesma que contém o programa gráfico. Pode ser o caso dos dois computadores estarem conectados em rede e, nesse caso, o computador que gera os comandos é chamado servidor, enquanto o que recebe e executa os comandos de pintura é chamado de cliente. O formato da transmissão desses comandos (chamado *protocolo*) também é padronizado, então é possível que duas máquinas com sistemas operacionais e *hardware* diferentes se comuniquem dessa forma. Se o OpenGL não está sendo executado numa rede, o computador é considerado ao mesmo tempo cliente e servidor.

Este tipo de *design* tem uma conseqüência muito importante: os comandos do OpenGL não são imediatamente enviados para o *hardware*. Primeiro, são agrupados para então serem enviados – otimizando o uso da rede e também oportunizando outras otimizações. Por conseqüência, um erro comum de muitos programadores é tentar medir o tempo levado pelos comandos de pintura

simplesmente adicionando funções antes e depois de um comando (ou conjunto de comandos) o que, nessa arquitetura, certamente gerará resultados inválidos.

Desta forma, não há garantias de que os comandos serão imediatamente executados, o que deve ser medido é a velocidade que o OpenGL armazena os comandos para futura execução. Forçar a execução dos comandos também é uma estratégia ruim, visto que muitas otimizações são simplesmente desprezadas. Nessa arquitetura, a forma mais confiável de se medir a performance é através de *software* especial (o qual pode ser feito *download* na página do OpenGL).

Recursos

Entre os recursos gráficos disponíveis pelo OpenGL, podem ser destacados os seguintes:

- Algumas formas de desenho de pontos;
- Ajuste de largura de linhas;
- Aplicação de transparência;
- Ativação/desativação de serrilhamento (*aliasing*);
- Mapeamento de superfícies com textura;
- Seleção de janela de desenho;
- Manipulação de fontes/tipos de iluminação e sombreamento;
- Transformação de sistemas de coordenadas;
- Transformações em perspectiva;
- Combinação de imagens (*blending*);

O OpenGL fornece um conjunto de comandos poderosos, mas primitivos. Portanto, todas as rotinas de desenho de alto nível devem ser elaboradas em função destes comandos. Sendo assim, foram desenvolvidas algumas bibliotecas para simplificar a tarefa de programação. Estas bibliotecas são apresentadas a seguir.

GLU - *OpenGL Utility Library*: contém várias rotinas que utilizam os comandos OpenGL de baixo nível para executar tarefas como, por exemplo, definir as matrizes para projeção e orientação da visualização e fazer o *rendering* de uma superfície. Esta biblioteca é fornecida como parte de cada implementação de OpenGL e suas funções usam o prefixo **glu**;

GLUT - *OpenGL Utility Toolkit*: é um *toolkit* independente de plataforma, que inclui alguns elementos GUI (*Graphical User Interface*), tais como menus *pop-up* e suporte para *joystick*. Esta biblioteca, escrita por Mark Kilgard, não é domínio público, mas é *free*. O seu principal objetivo é esconder a complexidade das APIs dos diferentes sistemas de janelas. As funções desta biblioteca usam o prefixo **glut**. É interessante comentar que a glut substituiu a GLAUX, uma biblioteca auxiliar OpenGL que havia sido criada para facilitar o aprendizado e a elaboração de programas OpenGL independente do ambiente de programação (*Linux*, *Windows*, etc.);

GLX - *OpenGL Extension to the X Window System*: fornecido como um *anexo* de OpenGL para máquinas que usam o *X Window System*. Funções GLX usam o prefixo **glX**. Para *Microsoft Windows* 95/98/NT, as funções WGL fornecem as janelas para a interface OpenGL. Todas as funções WGL usam o prefixo **wgl**. Para IBM/OS2, a PGL é a *Presentation Manager* para a interface OpenGL e suas funções usam o prefixo **pgl**. Para *Apple*, a AGL é a interface para sistemas que suportam OpenGL e as funções AGL usam o prefixo **agl**;

FSG - *Fahrenheit Scene Graph*: é um *toolkit* orientado à objetos e baseado em OpenGL, que fornece objetos e métodos para a criação de aplicações gráficas 3D interativas. FSG, que foi escrito em C++ e é separado de OpenGL, fornece componentes de alto nível para criação e edição de cenas 3D e a habilidade de trocar dados em outros formatos gráficos.

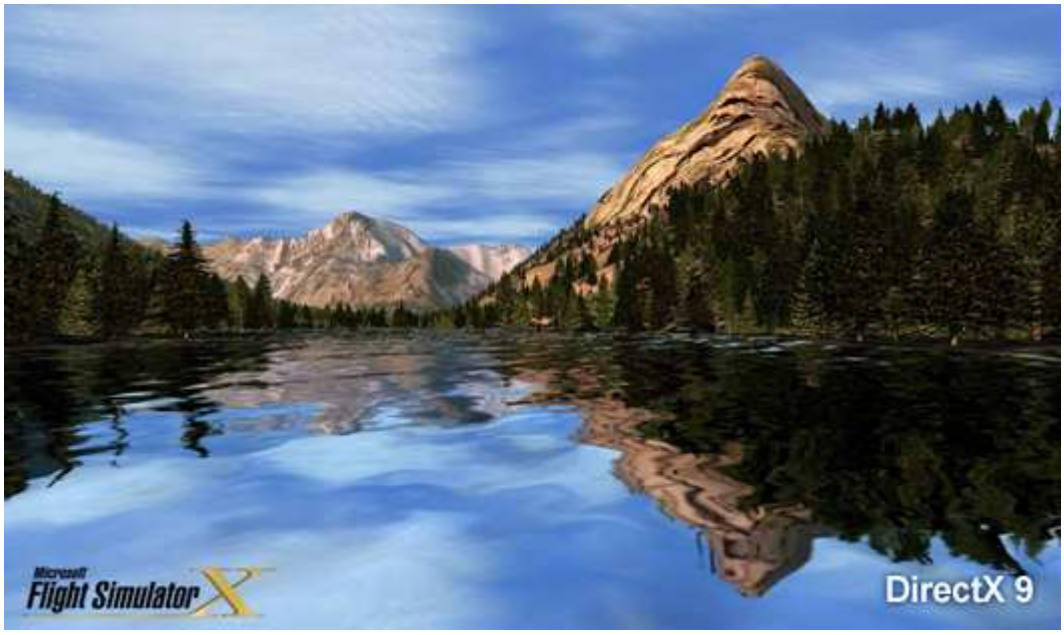
3.2 DirectX

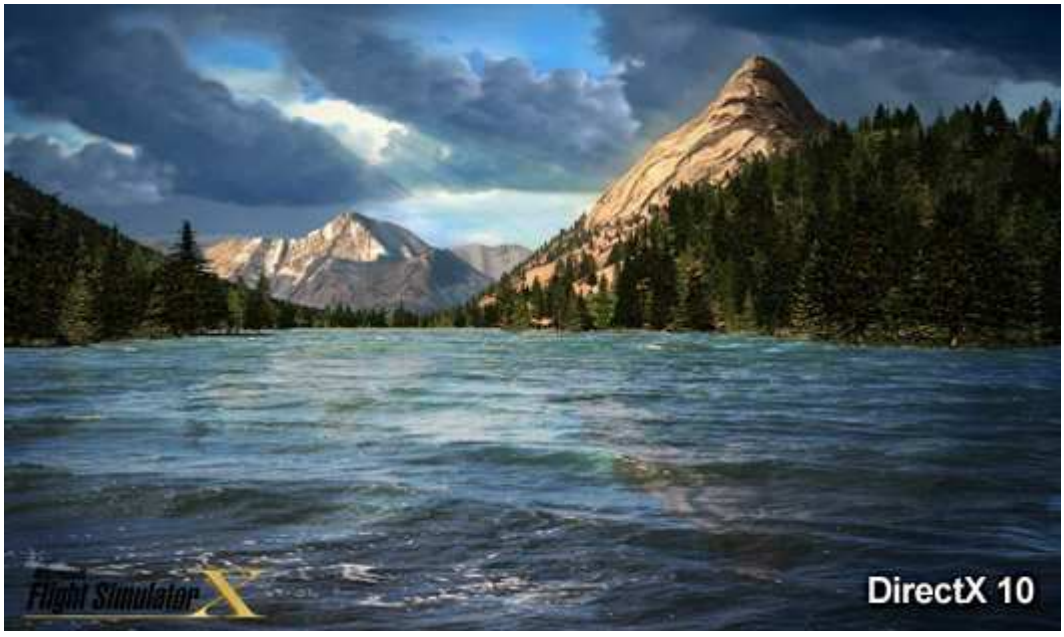
A primeira versão de DirectX, anteriormente conhecido como Game SDK, foi lançado em setembro de 1995. Seu objetivo era incentivar o desenvolvimento de jogos para o Windows, por isso chegou atrasado ao mercado. Antes do advento DirectX, a maioria dos jogos de computador eram escritos em MS-DOS, mas com a necessidade de se redesenhar a tela rapidamente para a animação em tempo real, obtendo assim velocidade satisfatória para animações, surgiu então esta coleção de APIs. O DirectX trata de tarefas relacionadas a programação de jogos e aplicações 3D para o Windows (exclusivamente).

O DirectX é um conjunto de interfaces de programação de baixo nível que oferece suporte de multimídia acelerada por hardware de alto desempenho aos programas Windows. O DirectX é usado para a comunicação entre a placa de vídeo e o computador, aprimorando os recursos multimídia do computador.

O DirectX permite que o programa determine os recursos de *hardware* do computador e defina os parâmetros do programa que devem ser correspondidos. Isso permite que os programas de *software* multimídia sejam executados em qualquer computador baseado no Windows com *hardwares* e drivers compatíveis com o DirectX e assegura que os programas de multimídia tirem proveito do *hardware* de alto desempenho.

Conforme o DirectX foi evoluindo os gráficos foram ficando cada vez melhores e com aspecto de real. As figuras abaixo retratam o mesmo cenário do jogo Flight Simulator em DirectX 9 e DirectX 10.





(Este texto ainda está em desenvolvimento)