7
Referências

ABNT. **ABNT NBR 15606-2:2007 Televisão Digital Terrestre- Codificação de dados e especificações de transmissão para radiodifusão digital. Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações**. ABNT - Associação Brasileira de Normas Técnicas. Rio de Janeiro, RJ. 2007. (ISBN: 978-85-07-00583-4).

ADOBE. ActionScript 3.0 overview. **Adobe**, 2000. Disponivel em: <http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html>. Acesso em: 19 Setembro 2010.

ADOBE. Adobe Flash Player. **Flash Player**, 2010. Disponivel em: <http://www.adobe.com/products/flashplayer/>. Acesso em: 18 Julho 2010.

ARIB. ARIB STD-B24, Version 3.2, Volume 3: Data Coding and Transmission Specification for Digital Broadcasting. **ARIB Standard**, 2002.

ATSC. DTV Application Software Environment Level 1 (DASE-1) PART 2:Declarative Applications and Environment, 2003.

BERNERS-LEE, T.; CAILLIAU, R. WorldWideWeb: Proposal for a HyperText Project. **WorldWideWeb:** Proposal for a HyperText Project., Geneva, 1990. Disponivel em: <http://www.w3.org/Proposal.html>. Acesso em: 18 Julho 2010.

BULTERMAN, D. C. A. et al. **AMBULANT: A Fast, Multi-Platform Open Source SMIL Player**. Amsterdam. 2004.

COSTA, R. M. D. R. Synchronization Management in DTV Applications, Rio de Janeiro, 2008.

COSTA, R. M. R. et al. Live Editing of Hypermedia Documents. **VII ACM Symposium on Document Engineering - DocEng2006**, Amsterdam, The Netherlands, p. 165-172, Outubro 2006.

DIRECTFB. **DirectFB**, 2010. Disponivel em: <http://www.directfb.org/>. Acesso em: Fevereiro 2010.

ECMA. Standard ECMA-262 : ECMAScript Language Specification. **Ecma International - Standard ECMA-262**, 2009. Disponivel em: <http://www.ecma-international.org/publications/standards/ecma-262.htm>. Acesso em: 19 Julho 2010.

ETSI. Multimedia Home Platform (MHP). **ETSI. Digital Video Broadcasting (DVB)**, 2003. Disponivel em: <http://www.mhp.org>. Acesso em: 20 Julho 2010.

FREEIMAGE. FreeImage. **FreeImage**, 2010. Disponivel em: <FreeImage>. Acesso em: 21 Agosto 2010.

GOMES, L. F. S. et al. **Nested Context Language 3.0 Part 9 – NCL Live Editing Commands**. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2006.

IETF. Real Time Streaming Protocol (RTSP). **IETF**, 1998. Disponivel em: <http://www.ietf.org/rfc/rfc2326.txt>. Acesso em: 26 Setembro 2010.

ITU. **ITU-T Recommendation H.761: Nested Context Language (NCL) and Ginga-NCL for IPTV Services.** Geneva. 2009.

JANSEN, J.; BULTERMAN, D. C. A. **Enabling Adaptive Time-based Web Applications with SMIL State**. Amsterdam. 2009.

KHRONA. Awesomium. **Awesomium Site**, 2010. Disponivel em: <http://www.khrona.com/products/awesomium/>. Acesso em: 20 Agosto 2010.

MICROSOFT. About Processes and Threads. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/ms681917(v=VS.85).aspx>. Acesso em: 16 Maio 2010.

MICROSOFT. Component Object Model. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/ms680573(v=VS.85).aspx>. Acesso em: 16 Maio 2010.

MICROSOFT. DirectDraw. **MSDN**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/aa917136.aspx>. Acesso em: 25 Agosto 2010.

MICROSOFT. DirectShow System Overview. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd375470(VS.85).aspx>. Acesso em: 12 Maio 2010.

MICROSOFT. Flipping Surfaces (Direct3D 9). **MSDN**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/bb173393%28VS.85%29.aspx>. Acesso em: 20 Agosto 2010.

MICROSOFT. How to Write a Source Filter for DirectShow. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd757807(VS.85).aspx>. Acesso em: 16 Maio 2010.

MICROSOFT. Overlay Mixer Filter. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd390946(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. Supported Formats in DirectShow. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd407173(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. Video Mixing Renderer Filter 7. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd407343(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. Video Mixing Renderer Filter 9. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd407344(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. Video Renderer Filter. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd407349(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. VMR Filter Components. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd390955(v=VS.85).aspx>. Acesso em: 15 Maio 2010.

MICROSOFT. Writing Transform Filters. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd391015(v=VS.85).aspx>. Acesso em: 16 Maio 2010.

MONOGRAM. MONOGRAM GraphStudio. **MONOGRAM**, 2009. Disponivel em: <http://blog.monogram.sk/janos/tools/monogram-graphstudio/>. Acesso em: 15 Maio 2010.

MORENO, M. F. **Conciliando Flexibilidade e Eficiência no Desenvolvimento do Ambiente Declarativo Ginga-NCL**. Rio de Janeiro: PUC-Rio, 2010. (Tese de doutorado a ser apresentada em 16 de Agosto de 2010).

MOZILLA. Gecko SDK. **Mozilla Developer Center**, 2010. Disponivel em: <https://developer.mozilla.org/en/Gecko>. Acesso em: 10 Agosto 2010.

MOZILLA. MDC Doc Center. **JavaScript**, 2010. Disponivel em: <https://developer.mozilla.org/en/JavaScript>. Acesso em: 10 Agosto 2010.

MOZILLA. Mozilla. **Mozilla Firefox**, 2010. Disponivel em: <http://www.mozilla.com/pt-BR/firefox/>. Acesso em: 20 Agosto 2010.

NETSCAPE. Netscape annouces plans to make next-generation communicador source code available free on the net. **Netscape News**, 1998. Disponivel em:

<http://web.archive.org/web/20021001071727/wp.netscape.com/newsref/pr/newsrelease558.html>. Acesso em: 23 Maio 2010.

OVERVIEW of Data Flow in DirectShow. **Site da Microsoft Developer Network**, 2010. Disponivel em: <http://msdn.microsoft.com/en-us/library/dd390948(v=VS.85).aspx>. Acesso em: 16 Maio 2010.

SANT'ANNA, F. F. G.; SOARES, L. F. G.; CERQUEIRA, R. F. D. G. **Nested Context Language Part 10 - Imperative Objects in NCL: The NCLua Scripting Language.** Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2008.

SOARES NETO, C. S. et al. **Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0**. Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, p. 223. 2010. 2a Edição, revisão 3.

SOARES, L. F. G. **Nested Context Language Part 11 – Declarative Objects in NCL: Nesting Objects with NCL Code in NCL Documents**. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2009.

SOARES, L. F. G.; BARBOSA, S. D. J. **Programando em NCL 3.0:** Desenvolvimento de Aplicações Para o Middleware Ginga. 1ª Edição. ed. Rio de Janeiro: Campus, 2009.

SOARES, L. F. G.; RODRIGUES, R. F. **Nested Context Model 3.0 Part 1 – NCM Core.** Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2005. (0103-9741).

SOARES, L. F. G.; RODRIGUES, R. F. **Nested Context Model 3.0. Part 1 - NCM Core**. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2005. (0103-9741).

SOARES, L. F. G.; RODRIGUES, R. F. **Nested Context Language Part 8 – NCL Digital TV Profiles**. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Rio de Janeiro. 2006.

THE WebKit Open Source Project. **Webkit**, 2010. Disponivel em: <http://webkit.org/>. Acesso em: 22 Setembro 2010.

THORN, A. **DirectX 9 graphics:** the definitive guide to Direct3D. Texas: Wordware Publishing, Inc., 2005.

W3C. XML Path Language (XPath) version 1.0. **XML Path Language (XPath)**, 2000. Disponivel em: <http://www.w3.org/TR/xpath/>. Acesso em: 20 Setembro 2010.

W3C. Scalable Vector Graphics (SVG) 1.1 Specification. **W3C**, 2003. Disponivel em: <http://www.w3.org/TR/2003/REC-SVG11-20030114/>. Acesso em: 21 Setembro 2010.

W3C. Document Object Model (DOM) Bindings. **W3C - World-Wide Web Consortium.**, 2004. Disponivel em: <http://www.w3.org/DOM/Bindings>. Acesso em: 28 Agosto 2010.

W3C. Document Object Model (DOM). **W3C - World-Wide Web Consortium.**, 2005. Disponivel em: <http://www.w3.org/DOM/>. Acesso em: 18 Julho 2010.

W3C. SMIL 2.0: Interactive Multimedia for Web and Mobile Devices., Abril 2005. Disponivel em: <http://www.w3.org/TR/REC-smil>. Acesso em: 22 Agosto 2010.

W3C. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**, 2008. Disponivel em: <http://www.w3.org/TR/2008/REC-xml-20081126/>. Acesso em: Agosto 2010.

W3C. XForms 1.1. **XForms 1.1**, 2009. Disponivel em: <http://www.w3.org/TR/xforms/>. Acesso em: 13 Agosto 2010.

W3C. HTML 5. **W3C**, 2010. Disponivel em: <http://www.w3.org/TR/html5/>. Acesso em: Agosto 20 2010.

W3C. HTML 5.0 Specification. W3C Recommendation. **W3C - World-Wide Web Consortium.**, 2010. Disponivel em: <http://www.w3.org/TR/html5/>. Acesso em: 17 Julho 2010.

Apêndice A: DirectX

Este apêndice tem como objetivo apresentar o DirectX, visto que ele foi escolhido como substituto do *backend* gráfico adotado pela implementação de referencia do *middleware* declarativo Ginga-NCL. O DirectX é um kit de desenvolvimento de *software* (SDK) criado pela Microsoft para a autoria de jogos e aplicações multimídia de alto desempenho. O DirectX oferece um conjunto de interfaces de programação em baixo nível voltados para a criação de gráficos em múltiplas dimensões, reprodução multimídia e manipulação de dispositivos de E/S.

As interfaces de programação do DirectX são distribuídas na forma de 5 (cinco) bibliotecas agrupadas por propósito. A primeira, o Direct3D, cuida do aspecto visual da aplicação, ou seja, oferece suporte a criação e desenho de modelos em múltiplas dimensões. A segunda, o DirectShow[3], é responsável por prover o suporte ao *streaming* e apresentação multimídia. A terceira, o DirectInput, é responsável por lidar com dispositivos de E/S. A quarta, DirectSound, é encarregada de lidar com a manipulação de áudio em três dimensões. A quinta e última, DirectPlay, é atribuída a tarefa de fornecer suporte a múltiplos utilizadores. Por não se aplicarem ao trabalho proposto, as duas últimas bibliotecas não serão abordadas neste apêndice.

A.1
Direct3D

O Direct3D é uma API parte do DirectX que permite ao usuário criar e manipular artefatos gráficos (como imagens, vídeos e figuras geométricas) em múltiplas dimensões. No contexto de uma aplicação cliente do Direct3D, um dispositivo gráfico é acessado através de uma API específica, através da qual são realizadas operações de renderização, de acordo com suas capacidades, a serem exibidas em uma ou mais telas conectadas a ele.

Uma aplicação Direct3D deve atualizar constantemente a tela do dispositivo de saída com os artefatos gráficos, para que as transições e

[3] O DirectShow a partir da versão 9.0c passou a fazer parte do kit de desenvolvimento do Sistema operacional Windows.

animações pareçam suaves. Além disso, a aplicação cliente deve obter os dados provenientes dos dispositivos de E/S a fim de garantir que a interação com o usuário seja corretamente interpretada. A rotina que promove a atualização constante da tela de exibição e trata os eventos de entrada do usuário é chamada de *render loop* (*loop* de renderização) ou *Game loop*.

O Direct3D aplica o conceito de superfícies com o objetivo de permitir a renderização de artefatos gráficos. Uma superfície nada mais é que um *buffer* de *pixels* alocado em memória, que delimita um retângulo plano onde é possível realizar as operações de escrita e leitura dos dados (*pixels*). Por exemplo, é possível carregar o conteúdo de uma imagem decodificada direto para uma superfície, copiar um determinado segmento de uma superfície existente para outra, realizar cópias sucessivas de quadros decodificados de um objeto de mídia do tipo vídeo para uma determinada superfície de destino ou ainda usar operações específicas para desenho livre.

No processo de renderização, o Direct3D pode utilizar um mecanismo de alternância entre superfícies para acelerar e sincronizar a apresentação, evitando efeitos indesejáveis (como o *flickering*[4]), uma vez que a taxa de atualização do dispositivo gráfico é diferente da taxa do dispositivo de saída (monitor de vídeo). O *back buffering*, conforme ilustrado na Figura 45, é um mecanismo bastante utilizado com esse propósito, ele consiste na alocação prévia de superfícies, no qual uma das superfícies assume o estado momentâneo de *front buffer*, pois é efetivamente visível e reflete a tela do dispositivo de saída, já as demais superfícies assumem o estado momentâneo de *back buffers*, atuando como áreas de transferência não visíveis.

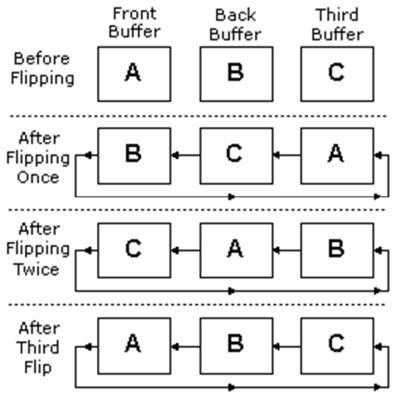

Figura 45 – Diagrama de *back buffering*. Retirada e adaptada de (MICROSOFT, 2010).

[4] Flickering é um efeito visual no qual a transição de quadros durante a exibição em uma tela de um monitor pode ser notada pelo olhar humano devido a interrupções frequentes.

Como mostra a Figura 45, no mecanismo de *back buffering* as superfícies são organizadas em correntes de troca (*swap chains)* e representadas por listas circulares, onde as cabeças são superfícies no estado *front buffer*. Em um primeiro momento, todos os artefatos gráficos a serem apresentados são transferidos para a superfície corrente, que se encontra no estado de *back buffer*, através da operação chamada de *rasterization*. Já em um segundo estágio, por meio de outra operação conhecida como *Page flipping*, a superfície corrente que está no estado *back buffer* assume o estado de *front buffer* e a superfície, que antes estava neste estado, passa para o estado de *back buffer*.

É interessante observar que o ciclo delimitado pelo momento em que os artefatos gráficos são copiados para a superfície que se encontra no estado de *back buffer* até o momento em que ocorre a migração de estado da superfície para o estado de *front buffer* e sua consequente apresentação é chamado de cena. Uma cena pode conter inúmeros artefatos gráficos dispostos segundo um espaço tridimensional definido. O Direct3D oferece estruturas em formato especifico para a especificação de artefatos gráficos.

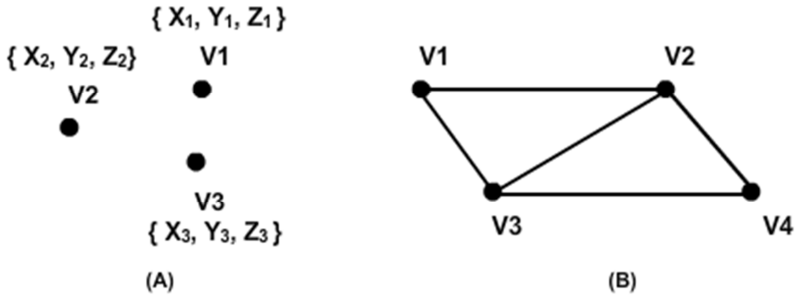

Figura 46 – (A) Vértices desconectados e (B) Polígono.

Como ilustra a Figura 46 (A), um ponto *V* qualquer (vértice) no espaço é representado por coordenadas no eixo x, y e z. Ainda na Figura 46 (A), a exibição dos vértices não traria nenhum benefício significativo ao aspecto visual da aplicação, por essa razão, o Direct3D permite a conexão entre os vértices para a composição de figuras geométricas primitivas mais elaboradas (como linhas, triângulos, quadriláteros e polígonos), conforme mostrado na Figura 46 (B). A união dessas figuras geométricas primitivas leva a formação de sólidos geométricos complexos (como cubos, esferas e cilindros), tal o qual mostrado na Figura 47.

Para definir o posicionamento de cada vértice que compõe uma figura geométrica primitiva no espaço definido pelo Direct3D, uma estrutura de dados, chamada de *vertex buffer*, é fornecida pelo SDK para guardar as informações

referentes às coordenadas. Contudo, como veremos a seguir, essa estrutura de dados não se limita a guardar informações referentes às coordenadas.

O uso de superfícies representa um grande avanço na apresentação de objetos de mídia, porém em uma cena do Direct3D, composta por sólidos geométricos (como cilindros, cubos e esferas), o uso de superfícies seria inviável, já que superfícies são planas e não se moldam ao artefato. Para tal, o Direct3D oferece uma abstração chamada de textura.

As texturas são um dos mais importantes recursos para aumentar o realismo em aplicações tridimensionais. Em suma, texturas são imagens planas de *bitmaps* desenhadas sobre a área de um ou mais polígonos (THORN, 2005, p. 131). Diferente de uma superfície comum, uma textura pode ser aplicada a área delimitada por um ou mais polígonos, ou seja, uma textura pode ser aplicada a todas as áreas delimitadas por uma figura não primitiva, como um cubo por exemplo.

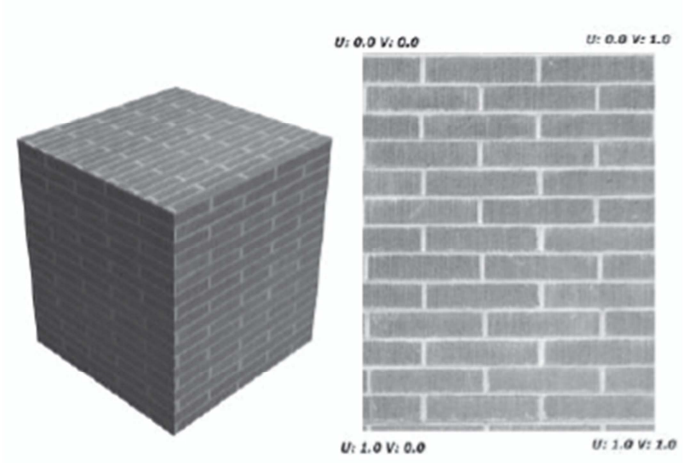

Figura 47 – Mapeamento de textura. Retirada de (THORN, 2005, p. 131).

Para que uma determinada textura seja aplicada a um polígono, em processo conhecido como mapeamento de textura, os vértices que compõe o polígono devem armazenar as informações para a orientação da textura, as chamadas coordenadas de textura, tal como ilustrado pelas letras *U* e *V* dispostas nas quinas do retângulo da Figura 47. O mapeamento de uma textura é feito de forma dinâmica, ou seja, no momento da rasterização dos polígonos é feita a associação entre a textura desejada e a figura geométrica alvo.

Os artefatos gráficos que compõe uma cena são apresentados de acordo com a posição determinada pelo observador principal. A precedência de visualização padrão dos artefatos gráficos é especificada pela ordem de renderização. No caso de aplicações mais exigentes, onde a ordem de

renderização não pode ser garantida, esse comportamento pode não ser desejável. Como por exemplo, a renderização concorrente de quadros de vídeo é um dos cenários onde a precedência de visualização não seria respeitada, visto que a todo instante ocorreria à sobreposição de quadros de vídeo de ambas as mídias.

Para tais casos, o Direct3D provê alguns mecanismos para garantia de precedência na apresentação, o *Z-Buffer* ou *Depth Buffer* é um deles. O mecanismo de *Z-Buffer* mantém um *buffer* com a distância de cada *pixel* para a câmera associada à visão do observador principal, fazendo uso dos valores (entre 0.0 e 1.0) da coordenada em z para determinar qual artefato gráfico deve sobrepor ao outro.

A.2 DirectShow

O DirectShow é uma biblioteca proprietária de uso livre da Microsoft que expõe uma API para manipulação de *streamings* multimídia em ambientes da família Microsoft Windows. A biblioteca permite que as aplicações clientes capturarem e reproduzam conteúdo audiovisual em alta-qualidade.

O DirectShow suporta vários tipos de dispositivos, arquivos e compressão de dados, que são distribuídos junto ao DirectShow ou junto ao sistema operacional Windows (MICROSOFT, 2010). Além disso, por adotar uma arquitetura modular aberta, qualquer usuário pode criar e distribuir seus próprios componentes. O que torna possível, a terceiros, expandir o suporte a qualquer dispositivo ou formato existente, desde que estejam em conformidade com a interface especificada.

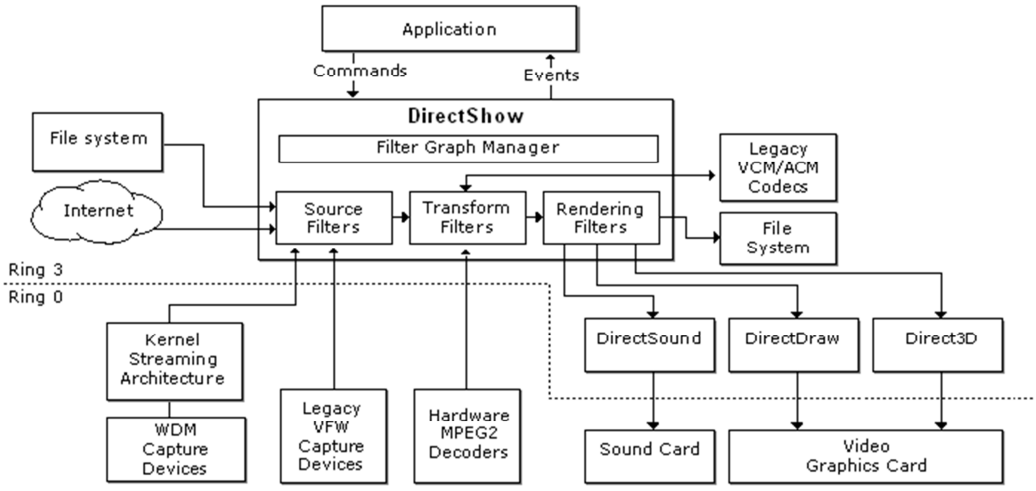

Figura 48 – Arquitetura do DirectShow. Retirada e adaptada de (MICROSOFT, 2010).

Conforme destaca a Figura 48, o modelo de representação da arquitetura do DirectShow estabelece que cada componente da arquitetura envolvido no processo de reprodução, desde a obtenção da *stream* até a sua renderização, seja representado por um filtro. Cada filtro representa a porção de software que desempenha um papel específico no processo. Um filtro se interliga a outros filtros por meio de arestas que se conectam a portas de saída ou entrada, os chamados pinos, formando uma cadeia de filtros, denominada Grafo de Filtros (*Filter Graph*).

Ainda na Figura 48, o Gerente de Grafo de Filtros (*Filter Graph Manager*) é o principal componente da arquitetura, sendo responsável por controlar os comandos invocados pela aplicação cliente, coordenar as mudanças de estado entre filtros, estabelecer um relógio de referência para os filtros, notificar as aplicações sobre eventos e prover métodos para a construção de Grafos de Filtros.

O fluxo de dados em um Grafo de Filtros (Overview of Data Flow in DirectShow, 2010) ocorre sempre em um único sentido, do filtro fonte para o filtro de renderização. Os dados de um determinado filtro são armazenados em *buffers*, onde cada buffer é encapsulado por um objeto COM (MICROSOFT, 2010) chamado de amostra de mídia (*media sample*). A cada pino de um filtro está associado um gerente de alocação de amostras (*allocador*), que fornece amostras de mídia ao filtro para que sejam preenchidas ou utilizadas.

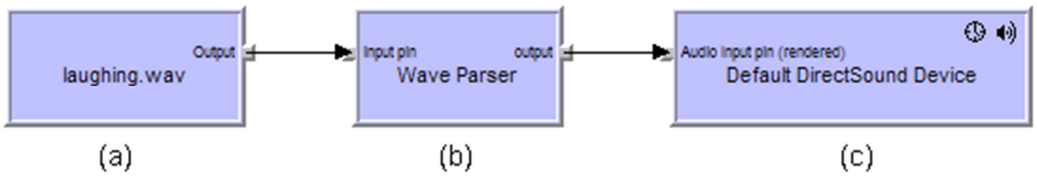

Figura 49 – Grafo de Filtros para reprodução de um arquivo no formato WAV. Retirado de (MONOGRAM, 2009).

Considerando o exemplo da Figura 49, uma vez criado o Grafo de Filtros e estabelecido o fluxo de dados, o filtro fonte em (a) requisita uma amostra de mídia ao gerente de alocação associado ao seu pino de saída (*Output*). De posse da amostra fornecida, o filtro fonte obtém o bloco de dados do dispositivo correspondente, preenche o *buffer* da amostra de mídia com os dados e a libera em seguida. A seguir, o filtro de transformação em (b), mediante notificação de liberação em (a), requisita ao pino de entrada (*input pin*) interligado ao pino (*Output*) em (a) a referência da amostra de mídia obtida. Ao mesmo tempo, ele requisita uma amostra de mídia livre ao seu pino de saída (*output*). Após o processamento de transformação de (b), os dados da amostra já processada são copiados para a amostra alocada junto ao pino de saída (*output*). Finalmente o filtro de renderização em (c) é notificado sobre a nova amostra, obtendo-a e repassando, nesse caso, ao dispositivo de áudio apropriado.

De acordo com a arquitetura apresentada na Figura 48, um Grafo de Filtros usual contém um filtro fonte, um ou mais filtros de transformação e outros filtros para a renderização. A classe de *filtros fonte* abstrai o acesso a um dispositivo de origem, que pode ser uma interface de rede, uma placa de captura de vídeo, um sintonizador de TV Digital ou uma unidade de armazenamento. O DirectShow oferece suporte a dispositivos de sintonização e captura baseados no WDM (*Windows Driver Model)* e a dispositivos de captura legados baseados no VFW(*Video for Window*). Inicialmente, um filtro fonte cria um thread de execução (MICROSOFT, 2010) cuja rotina envolve a obtenção de uma amostra de mídia associada ao pino de saída, o preenchimento da amostra com os dados obtidos do dispositivo, a marcação temporal da amostra e a entrega ao filtro de destino interligado (MICROSOFT, 2010).

A classe de *filtros de transformação* obtém os dados provenientes de um filtro fonte, processa-os com uma rotina definida via software ou por meio de um hardware específico. Nessa categoria, o filtro contém somente um pino de entrada e outro de saída (MICROSOFT, 2010), ambos de acordo com a especificação dos formatos de entrada e saída dos dados. Os codificadores e

decodificadores de áudio e vídeo são exemplos de filtros de transformação que podem ser introduzidos em um grafo de filtros.

A classe de *filtros de renderização* se posiciona ao final da cadeia de filtros, recebendo e apresentando os dados ao usuário. Dessa forma, um filtro de renderização de vídeo desenha os quadros na tela; assim como um filtro de renderização de áudio envia a unidade de áudio para o dispositivo de som; e um filtro de renderização de arquivos escreve os dados em arquivo. Essa classe de filtros será abortada com maior ênfase nesta seção devido a sua relevância para o trabalho proposto.

Por padrão, a Microsoft disponibiliza alguns filtros para a renderização de vídeo, como o VMR7 (MICROSOFT, 2010) e o VMR9 (MICROSOFT, 2010) em substituição aos respectivos filtros OMF (MICROSOFT, 2010) e o VRF (MICROSOFT, 2010). A partir da versão Vista do Microsoft Windows, um novo filtro EVR (Enhaced Video Rendering) foi criado, porém algumas limitações de plataforma e recursos o tornam inadequado ao escopo deste trabalho.

O filtro de renderização VMR representa um grande avanço na reprodução de vídeos para a plataforma Windows. Com a promessa de alto desempenho, a possibilidade de integração com motores gráficos em 3D e a compatibilidade com aplicações legadas, o VMR se apresenta como uma opção para a reprodução de vídeos em jogos 3D e aplicações multimídia. Em sua primeira versão, o VMR7, o filtro utiliza a API disponibilizada pelo *DirectDraw* (MICROSOFT, 2010) para realizar a renderização dos quadros decodificados do vídeo, se limitando a prover suporte a renderização acelerada somente em duas dimensões. Já na versão seguinte, o VMR9, o filtro utiliza a API do *Direct3D* para prover suporte a renderização acelerada em duas ou três dimensões.

Como destaca a Figura 50, a abordagem modular adotada na arquitetura de componentes do filtro VMR, permite que aplicações clientes configurem-no de várias maneiras, através da remoção, adição e customização de seus subcomponentes.

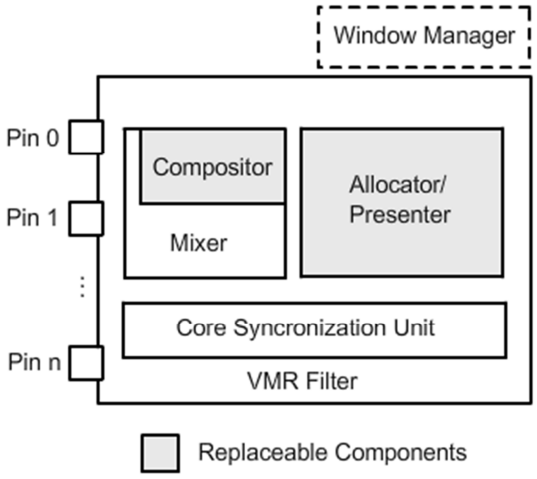

Figura 50 – Componentes do filtro de renderização VMR. Retirada e adaptada de (MICROSOFT, 2010).

Ainda de acordo com a Figura 50, um filtro VMR possui 5 (cinco) subcomponentes. O primeiro, o Core *Syncronization Unit*, garante que a marca de tempo das amostras de mídia sejam respeitadas. O segundo, o *Mixer*, é responsável por juntar várias *streams*, quando na presença de múltiplas *streams* de entrada, e garantir a ordem de superposição entre elas (*mixing mode*). O terceiro, o *Compositor*, é responsável pela execução das operações de *blending* sobre as *streams* de entrada em uma superfície *DirectDraw* ou *Direct3D*. O quarto, o *Allocator/Presenter*, é responsável por alocar amostras de mídias e fornecer objetos de apresentação (*DirectDraw* ou *Direct3D*), além de cuidar da comunicação com o dispositivo gráfico associado. Por fim, o quinto subcomponente, o *Window Manager*, é responsável por gerenciar as janelas que exercem um papel de container da área de renderização.

De acordo com a forma de renderização escolhida, um filtro VMR pode conter de dois a cinco subcomponentes, além dos pinos para a entrada de *streams*. O arranjo dos subcomponentes determina os modos de apresentação do filtro, sendo três os possíveis: *Windowed*, *Windowless* e *Renderless*.

No modo de apresentação *Windowed*, também chamado de modo de compatibilidade, a área de renderização do vídeo fica contida em uma janela criada pelo próprio filtro VMR. Considerando somente uma *stream*, o filtro carrega três subcomponentes próprios, sendo eles: o *Allocator/Presenter*, o *Core Syncronization Unit* e o *Window Manager*.

No modo de apresentação *Windowless*, ao contrário do modo *Windowed*, a área de renderização do vídeo não fica contida em uma janela própria, mas sim em uma sub-região da janela (*client area*) oferecida pela aplicação cliente.

Dessa forma, para uma *stream* como entrada, o filtro carrega dois subcomponentes, sendo eles: o *Allocator/Presenter* e o *Core Syncronization Unit*.

No modo de apresentação *Renderless*, a aplicação cliente deve criar a área de renderização, além de disponibilizar os procedimentos necessários para isso. Para uma *stream* como entrada, o filtro carrega dois subcomponentes, o primeiro, o *Core Syncronization Unit*, é fornecido pelo filtro, já o segundo, o *Allocator/Presenter*, deve ser fornecido pela aplicação cliente. Esse modo de apresentação, em particular é muito útil no desenvolvimento de games e aplicações multimídia que exijam efeitos sofisticados de vídeo, pois permitem as aplicações criar e controlar suas próprias áreas de renderização (superfícies *DirectDraw* ou *Direct3D*).

Cabe salientar que a distinção física entre as categorias de filtros não é absoluta, pois um determinado filtro pode acumular várias funções ou mesmo distribuir funções de uma mesma categoria em vários filtros.

A.3
DirectInput

O DirectInput é a biblioteca proprietária de uso livre da Microsoft que especifica a interface com dispositivos de entrada (como mouse, teclado, joystick e outros controladores). A API fornecida por essa biblioteca permite a aplicação cliente obter dados de dispositivos de entrada mesmo quando eles estão em plano de fundo. Através do recurso de mapeamento de ações, aplicações podem requisitar dados sem a necessidade de saber que tipo de dispositivo está sendo usado para gerá-lo.

O DirectInput funciona utilizando o mecanismo de *pooling* nos dispositivos de entrada especificados. Uma aplicação que deseje receber dados de um dispositivo de entrada deve enumerar os dispositivos disponíveis e escolher quais dispositivos manipular. A representação dos dados provenientes dos dispositivos ocorre da mesma forma pra todos os dispositivos, o que permite generalizar o tratamento dos eventos vindo dos dispositivos de entrada.