# Manipulações diretas em geometria 3D

Harlen Costa Batagelo, Artur Duarte da Conceição Neto

UNICAMP – Universidade Estadual de Campinas
FEEC – Faculdade de Engenharia Elétrica e de Computação
DCA – Departamento de Computação e Automação
(E-mail: harlen@dca.fee.unicamp.br, artur@hst.com.br)

**Resumo**

*Este trabalho sintetiza uma série de artigos relacionados aos aspectos implementacionais de interfaces gráficas 3D. Os artigos correspondem aos trabalhos apresentados na forma de seminário no decorrer da disciplina IA369J (Tópicos em Engenharia de Computação VI – 2o. semestre/2000 – Prof. Dra. Wu-Shin Ting). São analisados três artigos sobre interações 3D através de dispositivos 2D, três artigos sobre determinação de parâmetros de controle em modelagem geométrica e dois artigos sobre arquiteturas de desenvolvimento de interfaces 3D de manipulação direta. Os comentários são apresentados na forma de um resumo estendido. A formalização matemática foi evitada, dando maior ênfase aos resultados alcançados pelo algoritmo na solução do problema proposto. Essa avaliação foi realizada de forma crítica, de acordo com as idéias discutidas no seminário.*

## 1. Interações 3D

O desenvolvimento de técnicas eficazes de interação 3D para dispositivos de entrada e saída em 2D tem sido um desafio clássico na computação gráfica. Embora atualmente esse problema já possa ser tratado de forma satisfatória com a utilização de dispositivos especiais de realidade virtual e/ou realidade aumentada, os usuários em geral só dispõem de dispositivos de entrada e saída 2D populares, tais como joysticks, “mice” e monitores. Isso ocorre por uma série de fatores: baixo custo dos dispositivos 2D, facilidade de uso, pouco esforço físico, tamanho físico dos dispositivos, dificuldade de utilizar dispositivos essencialmente 3D para realizar operações 2D precisas, entre outros [Kettner].

Neste capítulo analisamos algumas soluções de hardware e software existentes para o problema de mapeamento 2D-3D. Embora as soluções por software sejam mais limitadas, são igualmente mais fáceis de se popularizarem devido ao baixo custo. A respeito dessas técnicas analisamos os trabalhos de [Nielson] e [Bier90]. Já as soluções por hardware, embora muito mais eficazes, são também muito mais difíceis de se consagrarem como dispositivos padrão. Ainda não há, por exemplo, um dispositivo de entrada 3D que seja tão bem aceito e difundido como um dispositivo 2D como o mouse. Sobre essa ótica analisamos o trabalho de [Balakrishnan] que sugere um novo dispositivo de entrada 3D semelhante ao mouse 2D e que herda suas vantagens: o “Rockin' Mouse”.

### 1.1 Direct manipulation techniques for 3D objects using 2D locator devices [Nielson86]

Neste trabalho, Nielson e Olsen descrevem técnicas de manipulação direta para tarefas de interação 3D usando apenas um dispositivo de localização 2D e uma visão ortográfica ou perspectiva. Em particular, descrevem técnicas para especificação de pontos 3D, translação, rotação sobre um eixo qualquer e transformações de escala. Por exemplo, para a especificação de um ponto 3D, utilizam-se de uma “tríade”, equivalente a um cursor 3D, visualizada como a projeção de três segmentos de linha perpendiculares entre si que representam uma base ortogonal. Cada eixo pode ser desenhado com uma cor diferente para permitir uma melhor distinção, e/ou flechas que indiquem a direção positiva de cada eixo. A localização dessa tríade é facilitada com a visualização de um cubo envolvente.

Os autores sugerem que a tríade pode ser manipulada através de um vetor direção calculado pela mudança da posição do localizador desde a sua última posição. A partir disso calculam-se os ângulos compreendidos entre esse vetor direção obtido e os eixos projetados. O movimento é feito ao longo do eixo cujo ângulo entre o vetor direção é minimizado. Em outras palavras, isso equivale a dividir o espaço 2D em seis zonas de movimentação divididas pelas bissetrizes dos ângulos entre os eixos projetados da tríade.

Para resolver o problema do mapeamento de duas dimensões de controle em três dimensões, todas as transformações de translação, rotação e escala são realizadas com uma ou mais dimensões fixas. Por exemplo,

para realizar uma translação o usuário fixa uma aresta ou uma face e determina o vetor direção através de pontos colocados nesse objeto. O mesmo raciocínio é aplicado para as transformações de rotação e escala.

Embora essas técnicas permitam a manipulação direta de objetos 3D, ainda persistem várias limitações, em especial a questão de projetar a tríade na tela 2D, visto que dois eixos podem se tornar indistinguíveis caso o ângulo entre seus eixos projetados se aproxime de zero. De fato, sempre haverá uma distorção em algum dos eixos, por isso a eficácia só é garantida caso dois dos eixos projetados não se aproximem de uma certa condição de paralelismo. Por essa questão de distorção, o método descrito se faz especialmente útil em projeções ortográficas, onde a distorção é invariante em todas as posições

## 1.2 Snap-dragging in three dimensions [Bier90]

Snap-dragging é uma ferramenta de construção de objetos tridimensionais que une três técnicas de interação 3D: funções de gravidade, alinhamento de objetos e movimentação suave. Cada componente é aprimorado individualmente, resultando numa combinação que simplifica de maneira significativa a edição de uma cena numa única visão em perspectiva, usando apenas um mouse e um teclado.

A maioria dos editores 3D de manipulação direta utiliza funções de gravidade que ajudam o usuário a colocar pontos de objetos em vértices, arestas e superfícies. Entretanto, tais funções só atraem o cursor para um tipo de objeto de cada vez (Ex. vértices e arestas, mas não ambos), fazendo com que o usuário perca tempo efetuando troca de comandos para obter a função de gravidade desejada. A técnica de snap-dragging resolve esse problema ao utilizar três funções de gravidade ao mesmo tempo, numa ordem de preferência designada pelo usuário.

Snap-dragging também aprimora a maneira como são realizadas as transformações afins nos objetos editados. Nos sistemas comuns de manipulação direta os objetos são rotacionados (ou transladados) com a ajuda de controles que atualizam a transformação em passos discretos, normalmente através da movimentação do mouse ou de dials. Por outro lado, para obter rotações mais precisas - como, por exemplo, uma rotação que faça com que a face de um objeto coincida com a face de outro - outras ferramentas se fazem necessárias. Outra técnica de rotação/translação anterior ao snap-dragging é a então chamada “skitters and jacks” [Bier86], que consiste em primeiro selecionar os objetos a serem rotacionados, colocar um objeto especial chamado “jack” que indica o eixo de rotação e entrar o valor de rotação através de um menu. O “skitter” é um objeto análogo à posição do cursor, mas dentro da cena 3D, que é utilizado para colocar o jack e definir a posição inicial e final da rotação. O skitter é visualizado na cena como uma tríade de segmentos de linha perpendiculares entre si: um para cada eixo de rotação. Todas essas técnicas, porém, tem a desvantagem de tirar a atenção do usuário para a entrada de um valor de rotação, ou confundi-lo com um número excessivo de jacks na cena. A técnica de snap-dragging simplifica esse processo ao adotar somente um jack, chamado aqui de âncora, que trabalha em conjunto com objetos de alinhamento cujas interseções são calculadas automaticamente. Visto que as funções de gravidade são imediatamente agregadas a essas interseções, torna-se possível obter transformações precisas seguindo esses pontos com o skitter.

Na técnica de snap-dragging, a função de gravidade é utilizada em todas as operações de interação com a cena. Fazem parte dessas atividades os comandos utilizados para posicionar o skitter em um local determinado, adicionar novos segmentos de linha e aplicar transformações afins suaves. A medida que o mouse é movimentado, o skitter também é alterado e posicionado no objeto mais próximo que contenha a função de gravidade. Snap-dragging aprimora as funções de gravidade de modo que elas passam a funcionar tanto para os vértices, arestas e faces ao mesmo tempo. Se o skitter não está no domínio da gravidade do vértice, o programa vai tentar atraí-lo para uma aresta. Se não for possível, vai tentar atrair para uma face. Finalmente, se não for possível, o skitter vai ser atraído para um plano default paralelo à tela. Essas preferências na função de gravidade podem ser alteradas através de um menu.

A técnica de snap-dragging calcula em tempo real todas as interseções entre objetos de alinhamento. Três tipos de objetos são utilizados para esse fim: linhas, planos e esferas. Nessas interseções são automaticamente agregadas funções de gravidade, de modo que o usuário possa construir novos objetos simplesmente posicionando o skitter sobre um desses pontos (ou curvas). Desse modo a edição do objeto assemelhe-se a construção de um desenho 3D com a ajuda de réguas e compassos no espaço.

Todas as transformações de snap-dragging seguem o skitter. Como o skitter pode se restringir a vértices, arestas ou faces, os objetos que estão sendo transformados podem ser alterados de forma precisa. As transformações possíveis são: translação, rotação sobre um ponto e rotação sobre um eixo. Essas transformações são realizadas com a ajuda de uma âncora – um objeto que define um eixo de rotação e três planos de alinhamento.

Enfim, a técnica de snap-dragging consegue simplificar de forma substancial o trabalho de manipulação de objetos 3D ao incorporar funções de gravidade, alinhamento e movimentação suave em um só conjunto. Dessa forma o número de operações é reduzido, extingue-se a necessidade de utilizar várias janelas com perspectivas diferentes para a edição dos objetos, e a construção de objetos consegue transmitir melhor a analogia da utilização de réguas e compassos em 3D através do cálculo automático de interseções entre objetos de alinhamento.

Embora o snap-dragging tenha conseguido resolver de forma satisfatória o problema de definir transformações afins precisas, surge o novo problema de definir pontos isolados no espaço, visto que todas as construções se restringem a pontos de objetos já inseridos ou pontos de interseção entre objetos de alinhamento. Ademais, ainda que os objetos de alinhamento forneçam uma maneira elegante de criar objetos no espaço (tais como poliedros), essa construção exige do usuário um bom conhecimento de geometria descritiva, o que em geral não existe. Por exemplo, como determinar quais são os objetos de alinhamento necessários para construir um dodecaedro?

## 1.3 The Rockin' Mouse: integral 3D manipulation on a plane [Balakrishnan]

O Rockin' Mouse é um dispositivo de entrada de quatro graus de liberdade que possui a forma física de um mouse comum, exceto pela parte inferior que é arredondada de modo a permitir movimentos de inclinação. É justamente essa inclinação que confere dois graus extras de liberdade, tornando-o apto a manipulação de cenas 3D.

Um mouse comum só aceita movimentos planares, e assim, no caso de manipulações em 3D, isso faz com o usuário precise alternar constantemente entre os modos de movimentação nos planos XY, YZ ou ZX. É de se notar que o trabalho de Nielson e Olsen [Nielson86] procura resolver essa questão através de software, mas isso introduz distorções devido a projeção da tríade. Já o trabalho de Bier [Bier90] não permite movimentações livres em 3D, mas somente através de objetos de construção. Com o Rockin' Mouse, todas as dimensões podem ser controladas simultaneamente, de forma tão natural como nas transformações planares com um mouse 2D.

Por que não usar simplesmente um dos vários dispositivos de manipulação 3D com múltiplos graus de liberdade já existentes, ao invés de usar um mouse? A justificativa é que a maioria dos usuários de aplicativos 3D não trabalham exclusivamente em 3D, e os dispositivos de interação 3D não são, em geral, tão eficazes nas manipulações 2D como o mouse. Além disso, mesmo em aplicativos 3D há um grande número de interações 2D, seja para acessar menus, seja para entrar strings de texto ou valores escalares. O Rockin' Mouse procura resolver esse desequilíbrio, procurando unir as interações 2D e 3D num único dispositivo. Os movimentos 2D são realizados da mesma forma que com um mouse comum. Já os movimentos em 3D são feitos através da sua inclinação, porém sem a necessidade de movimentar o dispositivo pelo espaço, o que é comum em dispositivos 3D, que certamente causam uma maior fadiga.

Da mesma forma que um mouse comum, o Rockin' Mouse percebe sua movimentação através de uma superfície de operação. Porém, sua base curva permite que o mouse possa ser inclinado sobre os eixos X (direita-esquerda) e Z (perto-longe). A quantidade dessa inclinação é medida por sensores que assim permitem o controle de mais dois graus de liberdade. A curvatura sobre o eixo Z é maior que aquela sobre o eixo X de modo a acompanhar a forma do mouse, e o centro da base é plano de modo a aumentar a estabilidade. Isso faz com o dispositivo possa se restringir a dois graus de liberdade mesmo estando, na verdade, manipulando quatro.

Os autores conduziram experimentos de comparação da eficiência do Rockin' Mouse com um mouse comum, para a realização de tarefas de manipulação 3D. Como era esperado, o Rockin' Mouse obteve melhores resultados, de pelo menos 30% de redução de tempo e em alguns casos de até 50%. Infelizmente os autores não realizaram uma comparação com outros dispositivos de manipulação 3D, o que seria o mais desejável.

## 1.4 Referências

[Bier86] Eric. A. Bier, Skitters and jacks: interactive 3D positioning tools, *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (Chapel Hill, NC, October 23-24, 1986), ACM, New York, 1987, pp. 183-196.

[Bier90] Eric A. Bier, Snap-Dragging in Three Dimensions, 1990, pp. 193-204.

[Nielson86] Gregory M. Nielson and Dan R. Olsen, Direct manipulation techniques for 3D objects using 2D locator devices, *Proceedings of the 1986 workshop on Interactive 3D graphics* (October 23-24, 1986, Chapel Hill, NC USA) pp. 175-182.

[Balakrishnan] Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach and George Fitzmaurice, The Rockin' Mouse: Integral 3D Manipulation on a Plane,
http://reality.sgi.com/gordo_tor/papers/rockmouse/rb1.htm

[Kettner] Lutz Kettner, Theoretical Foundations of 3D-Metaphors,
http://www.inf.ethz.ch/personal/kettner/pub/3d-metaphors.workshop.html

## 2. Pontos de Controle

A modelagem de sólidos é realizada através da mudança dos diferentes parâmetros que definem os sólidos em questão. Por exemplo, uma esfera pode ter como pontos de controle o seu centro e o seu raio, ou um segmento de linha que define o diâmetro da esfera e um eixo de revolução de círculos. Entretanto, é comum que os parâmetros originais da função não sejam suficientemente flexíveis para permitir construções mais complexas. Por exemplo, para achatar uma esfera ou deformá-la livremente, são necessários pontos de controle adicionais. Afinal, uma esfera achatada não é mais uma esfera, isto é, não pode mais ser representada unicamente por um centro e um raio. Novos controles se fazem necessários e, principalmente, pontos de controle que se relacionem com o objeto de forma intuitiva para o usuário. É nesse espírito que os seguintes artigos se baseiam. Through-the-lens control [Gleicher92] preocupa-se com a questão de determinar os parâmetros da câmera através de restrições adicionadas na própria imagem. Já o trabalho de Snyder [Snyder95] descreve um algoritmo de composição interativa de objetos 3D suaves de modo que eles não se penetrem, facilitando desse modo a construção de cenas fisicamente coerentes e não deixando de fora a interatividade. Finalmente, o artigo de Singh [Singh98] trata do problema mais clássico de modelagem de sólidos, introduzindo uma nova forma de controle que substitui ou generaliza as bem-conhecidas treliças (lattices) de pontos de controle através de curvas equivalentes a arames utilizados pelos escultores na construção de seus modelos.

### 2.1 Through-the-lens camera control [Gleicher92]

A determinação dos parâmetros de uma câmera é uma questão essencial na área de animação e composição de imagens. A maioria das formulações de uma câmera são construídas sob um modelo de projeção perspectiva especificada por um centro focal, um plano de visão e um volume de clipping. Essa determinação pode ser feita de diferentes formas, mas a mais comum ainda é através da especificação de três vetores - LookAt, LookFrom e ViewUp. O problema com a parametrização da câmera é que nem sempre um tipo de parametrização vai ser adequado para todos os propósitos desejados. Por exemplo, a parametrização LookAt/LookFrom/VUp é útil quando um ponto do espaço permanece centralizado na imagem enquanto a câmera move. Mas outros tipos de controles, tais como a restrição de vários pontos da imagem, seriam extremamente difíceis de serem realizados, embora teoricamente possíveis. A técnica Through-the-lens camera control procura resolver esse problema fazendo com que os parâmetros da câmera sejam calculados ao longo do tempo, de forma interativa e de acordo com controles aplicados na própria imagem.

Through-the-lens camera control é um corpo de técnicas que permitem ao usuário manipular uma câmera virtual através da restrição de características na imagem vista através das lentes. Em outras palavras, ao invés de determinar os parâmetros da câmera de forma direta, o usuário controla ou restringe pontos da imagem, distâncias, direções, etc., e o algoritmo acomoda o modelo de câmera para esses pontos de controle. Isso é basicamente um problema de especificação inversa: dados pontos na tela, determinar os parâmetros da câmera. Infelizmente, essa determinação deve ser feita através da solução de um sistema de equações não-lineares, que pode não ter solução ou ter infinitas soluções. Para contornar esse problema, Through-the-lens camera control calcula as derivadas de tempo dos parâmetros da câmera de acordo com as derivadas de tempo obtidas dos controles. O problema é reduzido a uma equação diferencial ordinária de primeira ordem através de otimizações restringidas baseadas nos pontos da imagem. Dessa forma, os parâmetros da câmera vão sendo ajustados de acordo com variáveis tais como a velocidade com que os pontos de controle são movidos ou arrastados, ao invés de calcular todos os parâmetros de uma só vez, dada a posição final do ponto de controle. A desvantagem desse método é que ele não permite alterar imediatamente a câmera, dados pontos de controle independentes dos pontos de controle existentes. Mesmo assim, para uma cena que está sendo editada em

tempo real, onde os controles podem ser pegos e arrastados, esse modelo produz resultados bastante satisfatórios.

Na implementação do algoritmo, o autor incluiu os seguintes controles de restrição: posição de um ponto na tela, distância entre dois pontos na tela, orientação de dois pontos na imagem e razão entre duas distâncias do espaço da tela. Nota-se, entretanto, que vários outros controles podem ser criados. O autor mostra como o algoritmo pode ser utilizado para realizar animações interativas onde a câmera acompanha as restrições impostas, ou para a dedução da posição da câmera real baseada numa fotografia, entre outras aplicações.

## 2.2 An interactive tool for placing curved surfaces without interpenetration [Snyder95]

Estão compreendidas na área de modelagem geométrica as tarefas de criar modelos a partir de um conjunto de partes primitivas ou objetos básicos e montar essas partes da forma desejada. Por exemplo, para modelar o motor de um carro, é comum que o usuário construa primeiro as diversas partes que formam o motor (pistões, parafusos e porcas, etc) e só então monte-as para formar o conjunto. Essa construção normalmente requer que os objetos não se penetrem, pois é assim que se comportariam na realidade. Em graus maiores de complexidade, cada uma dessas partes poderia ser modelada de forma que não se comportasse como um objeto geométrico estático, mas sim com parâmetros que lhe garantissem um certo controle de plasticidade. Exemplo disso é o modelo de uma toalha que, após ter sido colocada sobre uma mesa, ajusta sua forma de maneira fisicamente coerente.

O trabalho de Snyder está focado na questão de colocar objetos rígidos, sólidos e curvos em configurações fisicamente plausíveis, e de forma interativa. Neste trabalho, Snyder só trabalha com um objeto ativo de cada vez. Assim, dada uma condição inicial, o objeto ativo deve alcançar uma condição de balanceamento semelhante àquelas encontradas no mundo real. Exemplos incluem a tarefa de colocar uma colher numa tigela, enchê-la com cereais, colocar um telefone no gancho, colocar cubos de gelo num copo, modelar uma pilha de frutas, etc.

Para possibilitar a interação, Snyder não adota um modelo físico rigoroso. Em especial, não calcula a posição do objeto com relação ao tempo – assume-se que os corpos não possuem velocidade. De certa forma isso é desejável, uma vez que modelos físicos muito reais podem dificultar a colocação dos objetos. Por exemplo, assumindo um modelo físico acurado, um castelo de cartas poderia ruir quando uma nova carta fosse adicionada, e uma colher colocada numa tigela poderia quicar várias vezes e sair do lugar antes de atingir uma posição de repouso.

Por questões de eficiência, o tempo exato onde os pontos de contato ocorrem não é realmente calculado. Ao invés disso, calculam-se somente as transições da posição do objeto em passos discretos. Forças e torques são calculados para cada estado, embora essas forças não sejam reais, mas variáveis utilizadas para determinar de forma simplificada a convergência do objeto a uma posição estável. Com essas limitações o algoritmo atinge taxas de interatividade mesmo quando o objeto tratado toca vários outros objetos em vários pontos de contato. O autor também adota uma nova representação da superfície: um híbrido de aproximação poligonal que permite uma rápida detecção dos pontos de contato que surgem, e uma superfície paramétrica onde esses pontos são refinados através de iteração numérica.

Como é de se esperar, o algoritmo pode produzir resultados indesejáveis em determinadas situações. Por exemplo, ao jogar uma esfera em outra imediatamente abaixo, o algoritmo pode convergir com a esfera balanceada exatamente no topo da outra. Nesse caso o problema precisa ser resolvido através da intervenção do usuário, aplicando uma força local para a execução de um próximo passo. Outras configurações dessa espécie podem ocorrer em situações comuns onde temos uma superfície de contato e não um conjunto de pontos. Isso ocorre, por exemplo, na colocação de um toro sobre um cone ou na colocação de uma tampa sobre uma chaleira. Nessas situações o algoritmo requer de duas a três intervenções do usuário (para modificar pseudo-forças ou parâmetros de detecção de colisão) até que a configuração ideal seja alcançada. Esse tipo de intervenção pode não ser tolerável em ambientes interativos de modelagem.

## 2.3 Wires: a geometric deformation technique [Singh98]

Para a deformação de formas livres, é comum a utilização de treliças (lattices) de pontos de controle envolvendo o objeto, onde uma função relaciona esses pontos da treliça com pontos correspondentes no objeto. A deformação do objeto é feita por manipulação direta através da modificação dos pontos de controle. A forma mais comum de treliças é a do paralelepípedo envolvente, visto que formas mais gerais são muito difíceis de serem criadas. Essa limitação dificulta muito a deformação de formas complexas, pois não é

possível inferir facilmente quais pontos de controle deverão ser modificados para corresponder à deformação desejada no objeto. Além disso, um controle mais refinado da superfície requer um maior número de pontos de controle, isto é, uma treliça mais densa. Além de dificultar ainda mais essa manipulação, há um aumento considerável do custo de processamento. O algoritmo Wires traz uma alternativa a esse método através de uma técnica de modelagem geométrica que utiliza curvas próprias para a definição da forma e deformação dos objetos. Essas curvas fazem uma analogia com as armações de arame utilizadas pelos escultores na modelagem de uma escultura.

De forma mais específica, Wires utiliza dois tipos de curvas: curvas "wire" que definem a geometria do objeto e curvas domínio (domain curves) que definem o domínio da deformação sobre um objeto. Curvas wire são definidas por uma tupla contendo componentes que definem a curva paramétrica de forma livre, uma curva de referência, um valor de escalonamento em torno da curva de referência, um valor definindo o raio de abrangência em torno da curva de referência e uma função densidade que atua como fator de atenuação da deformação a medida que os pontos se afastam da curva de referência e alcançam o raio de atuação. Nesta definição, as transformações de escalonamento, rotação e translação podem ser tratadas de forma separada, facilitando a refinação do modelo. As curvas domínio, por sua vez, são curvas de forma livre que existem para controlar qual parte do objeto será deformada. Em resumo, as curvas domínio dizem quais partes do objeto serão deformadas, enquanto as curvas wire dizem o quanto será essa deformação.

Outros parâmetros de controle desenvolvidos pelos autores são os localizadores (locators), que permitem uma mudança local dos atributos da curva. Localizadores são inseridos ao longo da curva e contém definições próprias de escalonamento, raio de influência e função densidade (normalmente esses valores são constantes em toda a curva). Os atributos são interpolados entre os localizadores consecutivos de modo a obter um transformação suave dessa deformação. Localizadores também podem ser utilizados para inserir torções numa curva através da rotação da curva wire em torno da curva de referência nos pontos desejados.

O autor mostra vários exemplos onde o algoritmo wires pode ser utilizado, incluindo animação facial, simulação de tecidos, "costura" de superfícies e, inclusive, na simulação de treliças de deformação de formas livres. O algoritmo foi também implementado com sucesso como um módulo para o conhecido editor Maya da Alias|wavefront.

### 2.4 Referências


[Gleicher92] Michael Gleicher and Andrew Witkin, Through-the-Lens Camera Control, *Computer Graphics*, 1992, pp. 331-340.

[Snyder95] John M. Snyder, An Interactive Tool for Placing Curved Surfaces without Interpenetration, *Computer Graphics*, 1995, pp. 209-218.

[Singh98] Karan Singh and Eugene Fiume, Wires: A Geometric Deformation Technique, *Computer Graphics*, 1998, pp. 405-414.


## 3. Arquitetura de desenvolvimento de interfaces 3D de manipulação direta

Arquiteturas de desenvolvimento de interfaces 3D de manipulação direta procuram simplificar o processo muitas vezes tedioso de desenvolvimento de aplicações 3D. Bibliotecas de baixo-nível como OpenGL, GLIDE, Direct3D, entre outras, possuem poucos recursos para automatizar o processo de criação de cenas e interação 3D. Mesmo as bibliotecas "retained mode" como o GLUT e D3DRM disponibilizam no máximo recursos básicos de ray-picking e hierarquia de objetos.

Arquiteturas dessa espécie também dizem respeito aos editores de componentes de interação: embora sejam comuns os editores de componentes de interação (widgets) para interfaces 2D, o mesmo não é verdadeiro para o caso de componentes de interação 3D, pois parece não haver um consenso sobre quais são as melhores formas de widgets em 3D que correspondem aos componentes análogos em 2D. Assim, procura-se desenvolver bibliotecas de componentes de interação para interfaces 3D que forneçam um conjunto rico e extensivo de objetos 3D com suporte a manipulação direta, possibilitando ao usuário a interação com os objetos na mesma tela em que eles aparecem, de forma semelhante como já acontece para objetos 2D.

Outra discussão em torno dessa área existe com relação à flexibilidade dessas arquiteturas. Tais bibliotecas devem ser flexíveis o suficiente para adotar diferentes representações de objetos e, possivelmente, determinadas formas de manipulação bastante específicas e não-triviais. Acreditamos que poucos avanços tenham ocorrido nessa área porque, ainda hoje, a grande maioria do desenvolvimento de aplicativos 3D é feita

somente sobre bibliotecas de baixo nível (Ex.: OpenGL, Direct3D). Arquiteturas 3D como as analisadas aqui tem sido quase que inteiramente utilizadas para prototipação de aplicativos comerciais, pequenas aplicações, ou em pesquisa, quando a arquitetura é basicamente utilizada para visualização e modificação dos dados de entrada ou saída. Por outro lado, observamos uma espécie de período de transição e, às vezes, de relutância (ainda com razão) como o que ocorreu durante a migração de linguagens de baixo nível para alto nível. É certo que arquiteturas 3D como as analisadas aqui substituirão as APIs de hoje, da mesma forma que essas APIs já substituíram o código de máquina bruto. Isso, entretanto, só acontecerá efetivamente a medida que a evolução da capacidade de processamento dos computadores suporte essa mudança. É claro, não há dúvidas que além disso prepondera um certo aspecto comportamental dos desenvolvedores que, num primeiro instante, são relutantes em aceitar ferramentas de alto nível (e alguns radicais, como aqueles que preferem C ao C++, assembly ao OpenGL, etc). Essa análise, entretanto, está fora do escopo deste trabalho.

### 3.1 An object-oriented 3D graphics toolkit [Strauss92]

Este trabalho apresenta um conjunto de ferramentas de rápido desenvolvimento de aplicativos 3D com manipulação direta. Também implementado com o nome Open Inventor, esse trabalho procura simplificar a tarefa de composição de cenas gráficas, salvando o desenvolvedor da implementação de tarefas não relevantes para o propósito final do trabalho. Isso é desejável porque é comum o desenvolvedor optar por simplificar ao máximo as tarefas de interação com a cena, visto que a implementação dessas tarefas consomem muito tempo de desenvolvimento. O Open Inventor roda sob OpenGL e C++, reunindo, dessa forma, aspectos de portabilidade, expansibilidade e suporte à orientação a objetos.

Como já discutimos acima, uma ferramenta de alto nível do porte do Open Inventor deve ser flexível o suficiente para dar suporte às necessidades mais gerais dos desenvolvedores de ambientes gráficos. Infelizmente não encontramos a plataforma ideal no Open Inventor, mas ao menos ela permite a expansão de suas classes, tornando-o flexível e adequável para as necessidades mais comuns do desenvolvedor.

O Open Inventor está baseado fundamentalmente numa base de dados que contém uma representação da cena 3D (o scene database). Representação esta feita através de um grafo, normalmente acíclico e direcionado. Cada elemento desse grafo (node) representa um determinado objeto, que pode ser uma forma geométrica (shape node), um atributo dessas formas (property node) ou um nó que conecta outros nós em grafos e subgrafos (group node). Também são disponíveis nós que representam câmeras e luzes. As ações realizadas sobre as formas geométricas também são tratadas como objetos. A aplicação de uma ação a uma determinada cena define um objeto que percorre o grafo em profundidade, da esquerda para a direita, para realizar operações específicas como renderizar a cena, calcular a caixa envoltória de um objeto ou grupo de objetos, realizar uma procura ou gravar a cena em arquivo. Esse comportamento pode ser especificado pelo próprio nó, impondo à ação um determinado modo de travessia do subgrafo subseqüente. Determinados nós do group node funcionam como separadores (separator nodes), que têm a propriedade de gravar o estado atual do grafo, restaurando-o após a passagem da ação. A utilização dessa separação é desejável quando ocorrem várias instâncias de um mesmo objeto na mesma cena, impedindo que a alteração de um deles implique na alterações dos outros. O Open Inventor também permite reunir características de uma cena em um subgrafo chamado node kit, que pode ser utilizado e repetido arbitrariamente na cena principal.

Outra ferramenta disponível no Open Inventor é o sensor. Por exemplo, um sensor de dados (data sensor) pode ser utilizado para disparar um evento (na verdade, uma função callback é chamada) assim que houver uma mudança na base de dados da cena. Da mesma forma, uma sensor de tempo (timer sensor), pode ser configurado para acionar um evento em intervalos regulares de tempo.

Um tratamento simples de eventos é utilizado pelo Open Inventor. Um evento específico do sistema de janelas é gerado como resultado de alguma ação do usuário (movimentação do mouse, digitação, etc.) e é passado para a instância da área de renderização que corresponde à janela onde ocorreu o evento. Se o componente utilizado não processar o evento, ele é convertido para um evento independente do sistema de janelas e distribuído pelos nós, para o caso de algum deles tratá-lo de maneira específica.

A flexibilidade do Open Inventor está resumida na adição de novos node kits e na chamada de funções durante o percurso do grafo. Cada nó pode conter uma chamada callback (callback node) que invoca uma função determinada quando este nó é percorrido, ou quando este nó está a ponto de ser alcançado, ou quando o nó acabou de ser alcançado. Isso permite introduzir comportamentos especializados dentro do grafo da cena e acessar o estado atual do percurso do grafo.

### 3.2 An architecture for an extensible 3D interface toolkit [Stevens94]

Neste trabalho é apresentada uma plataforma para criação e manipulação de primitivas 3D, implementação de interfaces e widgets, o qual permite manipulações diretas e podem ser ligadas através de uma linguagem visual formando primitivas mais complexas.

Existe um avanço considerável nesta área para plataformas de interface 2D, de modo que, neste caso, os estudos já estão bem consolidados. Entretanto, o mesmo ainda não ocorreu para o caso 3D, embora já tenham começado a surgir ferramentas para 3D que se baseiam na construção de objetos a partir de primitivas gráficas e restrições aplicadas as primitivas. Em especial, a ferramenta proposta neste trabalho foi projetada com os objetivos de que tanto programadores como não-programadores possam prototipar interfaces 3D, e que o projeto seja extensível.

A arquitetura é baseada na criação de vínculos que relacionam as propriedades de uma primitiva gráfica com as propriedades de outra primitiva gráfica as quais são criadas através de uma interface visual. Cada objeto gráfico é uma instância de uma classe que representa uma primitiva. Por sua vez, cada classe contém variáveis que representam as características da primitiva, chamadas de slots. Técnicas de interação especificam como modificar um slot mantendo as restrições estabelecidas por outros slots. Para obter e atribuir valores para um slot, é usado um método de pesquisa (inquiry) que calcula o valor de um slot sob restrições. Outros métodos disponíveis são: assignment, establish e resolution. O método de atribuições (assignment) determina como a restrição do slot é mapeado de volta para primitiva a qual ele está restrito. Os métodos de pesquisa e atribuição podem ser complexos chamando funções de outros slots. O método establish tem a função de satisfazer as restrições. O método resolution é definido para cada primitiva para resolver seus slots, atualizar sua representação gráfica e chamar o método resolution de todas as primitivas ligadas.

Como exemplos de primitivas básicas temos, para o ponto, o slot posição; para o vetor, os slots direção, posição e tamanho,

O encapsulamento estrutural ajuda a reproduzir um conjunto de primitivas encadeadas. O encapsulamento de classe ocorre quando o mesmo é associado a uma nova classe, que é como se estivéssemos acrescentando uma nova primitiva a plataforma  O encapsulamento parametrizado pode ser recriado a partir de um conjunto de primitivas usadas como parâmetro. Através dos limites pode ser inserida restrições que limitam as variáveis do slot a intervalos desejados.

Esse conjunto de ferramentas tem sido utilizada na prototipação rápida de ferramentas para modelagem mecânica, visualização científica, construção de componentes de interação 3D e ilustrações matemáticas. Tarefas simples são realizadas rapidamente através desta arquitetura, mas a utilização de poucas primitivas (basicamente: ponto, vetor, plano) podem não ser suficientes para tarefas mais complexas.

## 3.3 Referências

[Strauss92] Paul S. Strauss and Rick Carey, An Object-Oriented 3D Graphics Toolkit, 1992, pp. 341-347.

[Stevens94] Stevens, M.P., Zeleznik, R.C., Hughes, J.F., An Architecture for an Extensible 3D Interface Toolkit, *Proceedings of UIST '94*, ACM SIGGRAPH, 1994.