Departamento de Engenharia de Computação e Automação Industrial
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

Relatório Técnico
Technical Report
DCA-003/03

Extensão das Funções de Emulação do Cursor3D no MTK

Daniel Tost

Orientadora:
Profa. Wu, Shin - Ting

State University of Campinas)
FEEC (School of Electrical and Computer Engineering)
DCA (Dept. of Computer Engineering and Industrial Automation)
{tost,ting}@dca.fee.unicamp.br[2ex]

09 de Maio de 2003

Abstract

Neste relatório final de Projeto de Pesquisa, financiado pela Fapesp e realizado durante o período de maio/2002 a abril/2003 (Processo n^o 02/01162-0, nível de Iniciação Científica) apresentamos novas funcionalidades para o *Cursor 3D*, entre elas, uma nova forma de movimentação, restrita à geometria de superfícies arbitrárias definidas pelo aplicativo, além de um inovador processo de seleção tridimensional de elementos. Mostramos também uma reformulação na estrutura de nossa biblioteca, cuja finalidade é a adequação aos padrões de projeto existentes, visando uma melhor organização e maior facilidade para manutenções ou futuras implementações.

1 Objetivo e Motivação

Devido ao desenvolvimento tecnológico e a melhora dos *hardwares* gráficos, ocorre também um consequente aumento da utilização de cenas tridimensionais nas mais diversas áreas da computação. Por isso, muitas vezes precisamos de uma interface de manipulação gráfica que nos permita uma apresentação natural e intuitiva. Além disso, em muitas situações é interessante a utilização de um equipamento convencional para essa manipulação, evitando gastos dispensiosos e desnecessários.

Por todos estes motivos, foi incluída na biblioteca MTK a classe de *Cursor3D*, uma ferramenta que permite uma livre movimentação pelo espaço tridimensional, através da utilização de um dispositivo de entrada comum, o *mouse*. O MTK é uma biblioteca de suporte a aplicativos gráficos interativos 3D que adota o padrão OpenGL para acessar as funções gráficas tridimensionais.

Como proposta do nosso projeto, buscamos tornar o MTK uma biblioteca mais completa. Para isso, estaremos estendendo as funcionalidades providas pelo *Cursor3D*.

As alterações visam oferecer ao usuário uma maior interatividade com a cena tridimensional através de novas formas de manipulação dos elementos e superfícies pertencentes a esta. Desenvolvemos algoritmos para a implementação de uma seleção tridimensional de elementos e para restrição do movimento do *Cursor3D* às superfícies arbitráriamente definidas pelo usuário.

A parcela do trabalho referente a seleção tridimensional foi descrita detalhadamente em meu relatório anterior [10], que será sintetizado na seção 6.

Neste relatório iremos nos concentrar no movimento do cursor restrito às superfícies. Esta nova forma de movimentação permite uma maior precisão na manipulação de um ponto específico de uma superfície. As concepções e decisões do projeto deste tipo de movimentação serão descritas nas seções seguintes (seção 8.1). Explicitaremos os problemas encontrados durante sua fase de planejamento e implementação (seção 8.2), e apresentaremos, ainda, os resultados obtidos (seção 8.2.5).

Por completude, apresentamos nas seções 2 e 11 o plano e o cronograma deste trabalho.

2 Plano de Trabalho

Apresentamos nosso plano de trabalho integral.

1. Estudo da linguagem C++ e a interface OpenGL
2. Estudo do sistema de janelas X Windows
3. Familiarização com MTK e MTKX
4. Estudo do conceito do cursor 3D
5. Projeto do cursor 3D com movimento restrito as superfícies específicas de uma aplicação
6. Extensão das funcionalidades do cursor 3D no MTK
7. Projeto de um mecanismo de seleção em 3D por mouse 2D
8. Inclusão do mecanismo de seleção no MTK
9. Depuração e Testes
10. Análise e comparação de performance do cursor 3D controlado por dispositivo de entrada 3D e controlado por dispositivo de entrada 2D.
11. Elaboração da documentação

Após a conclusão do primeiro período referente ao nosso trabalho, foi feita, então, uma reavaliação do cronograma inicial e propusemos as seguintes atividades para a segunda fase do projeto

1. Projeto do **cursor** 3D com movimento restrito às superfícies específicas de uma aplicação. Tais superfícies não precisam ser implicitamente definidas.
2. Reestruturação das classes do MTK.
3. Depuração e testes.
4. Análise e comparação do desempenho do cursor 3D controlado por dispositivo de entrada 3D e controlado por dispositivo de entrada 2D.
5. Elaboração da documentação.

a serem executadas de acordo com o seguinte cronograma.

Ativ.	Nov–Dez	Jan–Fev	Mar–Abr
1	X	X	X
2	X		
3	X	X	X
4			X
5	X	X	X

3 Trabalhos Correlatos

- Manipulação de Cenas 3D

  Pierce, Forsberg, Conway, Hong, Zeletnik, Mine [1]: O trabalho apresenta novas técnicas de interação, a partir das quais, o usuário pode interagir com as projeções bidimensionais formadas pelos objetos tridimensionais no plano da tela. A diferença em relação ao nosso trabalho é a abordagem adotada, os usuários ficam imersos em um ambiente totalmente virtual e utilizam-se de dispositivos de entrada nada convencionais. As manipulações dos objetos na cena são realizadas através de grandezas físicas como pressões, forças e temperatura, algo semelhante ao visto no filme *Minority Report* [4].

  Interfaces hápticas [13]: Háptica é um estudo de como combinar movimentos do sentido humano do tato com interação em computadores, ela utiliza dispositivos manipuladores que interagem com músculos e tendões humanos gerando entradas para um sistema. É uma área muito pequisada nos Estados Unidos, especialmente na Northwestern University. Trata-se de uma abordagem, em interfaces de manipulação, totalmente diferente da utilizada em nosso trabalho, no entanto merece ser mencionada.

  Gleem [3]: É uma pequena biblioteca de *widgets* 3D inspirados no Open Inventor [7], desenvolvida por Kenneth B. Russell, e que suporta direta interação do usuário com a cena tridimensional. Pode ser incorporada a programas desenvolvidos em Java [5] ou C++ [6]. Esta biblioteca possui algumas limitações. Devido a opção pela portabilidade, não há o conceito de grafo de cena e também todos os manipuladores são sempre definidos nas coordenadas do mundo. A diferença entre as bibliotecas é que o MTK, também projetado com a portabilidade em mente, deixa a cargo da aplicação estabelecer a conexão entre um manipulador e os objetos manipulados por ele. No MTK, o comportamento geométrico de cada "parte" do manipulador é configurável. Além disso, é o objetivo deste projeto que o MTK proveja *cursor 3D* capaz de dar suporte a posicionamento e seleção em cenas tridimensionais.

  Conner, Snibbe, Herndon, Robbins, Zeletnik, van Dam [2]: Apresenta um sistema que permite experimentos com *widgets 3D*, promovendo uma integração entre esses *widgets* e os objetos da aplicação, permitindo uma interação entre aplicação e interface gráfica interativa bem maior do que a dos *toolkits* interface-usuário tradicionais.

  Abrantes [11]: Apresenta um trabalho sobre manipulação de ce-

nas 3D a partir de dispositivos de entrada tridimensionais, no caso o Labtec Spaceball 3003. Este dispositivo é usado dentro do MTK para a movimentação do *Cursor3D*(antes apenas provida pelo *mouse* convencional) e da câmera. Neste trabalho ele ainda mostra a utilização de dois dispositivos de entrada, um bidimensional e outro tridimensional, simultanêamente dentro do MTK

- Movimento e Navegação Restritos a Superfície

  Clifford [8]: Este trabalho apresenta uma maneira de realizar movimento sobre uma esfera qualquer. O autor utiliza o conceito do plano tangente e também define um plano J, denominado por ele "plano do grande círculo", que é definido por três pontos: o ponto do movimento sobre a esfera, o centro e o ponto do pólo norte da esfera. Dado um ponto sobre a esfera, um movimento a partir dele pode ser realizado sobre o plano tangente ou sobre o grande círculo dependendo de sua direção.

  Bier [14]: O *Snap-dragging* é uma ferramenta de construção de objetos tridimensionais baseada em funções de gravidade. Quando se está construíndo uma figura geométrica, perde-se muito tempo em tarefas de alinhamento de arestas, planos e posicionamento correto de partes do objeto. Devido a este problema o *Snap-dragging* prove funções de gravidade e construções auxiliares que aceleram o posicionamento das estruturas.

4 MTK

O MTK é uma biblioteca que encapsula recursos de visualização e interação de forma que seja possível construir uma interface gráfica com manipulações diretas 3D. Ele começou a ser desenvolvido em 1995 no contexto de um projeto de pesquisa financiado pela Fapesp (Processo n^o 1996/0962-0) e compreende na versão corrente 8 classes de objetos:

- mtkDisplayList: responsável pelos modelos gráficos 3D e suas características gráficas (cor, material, etc.).
- mtkCameras: responsável pelas transformações de visualização (projeção paralela e perspectiva).
- mtkLights: responsável pelas fontes luminosas que interagem com os modelos gráficos.
- mtkSelection: responsável pela seleção de objetos existentes num cenário.

- mtkGuides: responsável por melhorar a percepção de profundidade.
- mtkConstraints: responsável pelo mapeamento $f: \Re^2 \rightarrow \Re^3$.
- mtkDraggers: responsável pela realimentação visual das ações do usuário através de *draggers*.
- mtkCursor: responsável pela realimentação visual do movimento do mouse 3D.

Todo o trabalho apresentado neste relatório é uma forma de complementar e aperfeiçoar esta biblioteca.

A figura 1 sintetiza a arquitetura da versão do MTK no início deste projeto, apresentando as principais relações entre as 8 classes.

O método cameraPointer3D() representado na figura é responsável pelo controle das funcionalidades do Cursor 3D, como por exemplo movimentação e seleção tridimensional.

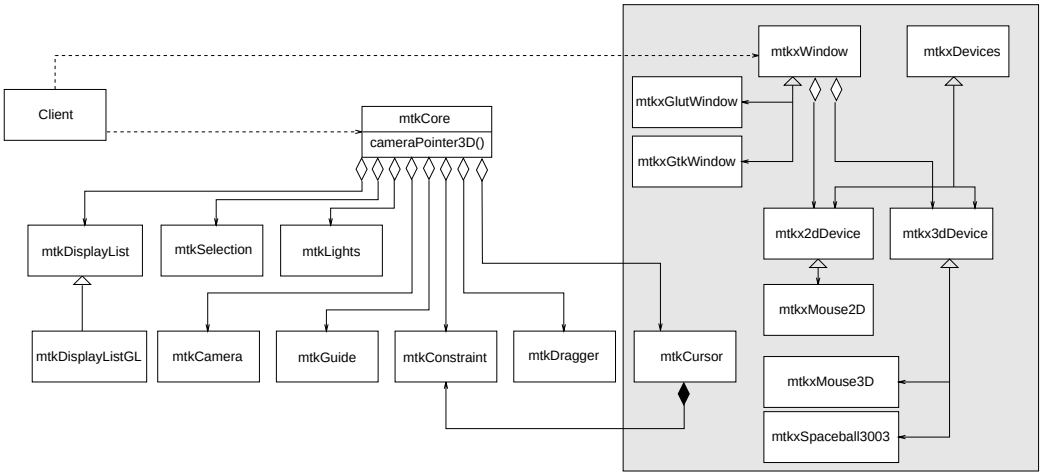


Figure 1: Arquitetura do MTK.

A seguir descreveremos o processo de seleção 2D e o cursor 3D disponível na última versão estável do MTK, eles são explicados mais detalhadamente em [10].

5 Seleção Bidimensional

A seleção bidimensional é um processo para identificação de elementos em uma cena tridimensional que considera apenas a área ocupada por estes no plano de projeção, em detrimento da coordenada referente a profundidade.

O MTK implementa este tipo de seleção, acrescentando a ele alguns conceitos adicionais como modos de seleção e de identificação de elementos [10].

Os objetos selecionados por este método são assocados a um ou mais *draggers*, através dos quais são manipulados bidimensionalmente no volume de interesse. O tratamento deles é deixado a cargo do aplicativo que deve especificar o procedimento desejado para esta tarefa.

6 Seleção Tridimensional

A seleção tridimensional é outro modo de identificação de elementos.

Nossa abordagem faz uso da definição de **Volume de Picking**, que é um volume cuja intersecção com os elementos da cena define o estado de seleção destes. A associação deste volume com o *Cursor 3D* provém uma interessante forma de manipulação e identificação de objetos.

Em nosso trabalho anterior desenvolvemos um algoritmo para realização deste tipo de seleção. Nosso algoritmo explora as características do *Depth Buffer* e a flexibilidade deste dentro do OpenGL. Dessa maneira ele não faz utilização de cálculos geométricos, procurando obter o melhor desempenho possível e prover uma interface de manipulação independente dos aplicativos.

O algoritmo é explicado detalhadamente em [10], algumas pequenas modificações ocorreram, fruto da reestruturação sofrida pela biblioteca (vide seção 7).

Conceitualmente, o nosso algoritmo de seleção 3D consiste de duas fases:

1. seleção bidimensional de todos os elementos no plano de projeção
2. seleção considerando a profundidade dos objetos.

Para a realização da primeira fase da seleção 3D o **Volume de Picking** é decomposto, a partir desta decomposição conseguimos uma área no plano de projeção sobre a qual será realizada a seleção bidimensional. É criada uma lista contendo todos os elementos que interceptam esta área, sobre a qual será realizada a segunda fase da seleção.

Em nosso trabalho anterior chamavamos a atenção para um modo de seleção especial criado exclusivamente para realizar esta tarefa, sem, no entanto, alterar o conjunto de elementos selecionados. Após a reestruturação do módulo de seleção do MTK, a necessidade deste modo deixa de existir. Foram definidos dois métodos para a seleção

tridimensional, um para cada uma das fases, a seleção bidimensional provida por um destes métodos por si própria não altera o conjunto de elementos selecionados.

Para a realização da segunda fase, o algoritmo aproveita-se das facilidades relacionadas ao *Depth Buffer*. Através de rerenderizações do *cursor 3D* e, um a um, dos objetos da lista retornada pela seleção bidimensional, o algoritmo busca descobrir a profundidade dos elementos e relacioná-las com a coordenada z do *cursor 3D* para obter o elemento que estiver mais próximo dele.

A comparação de profundidade entre os objetos projetados num *pixel* e a posição espacial do *cursor 3D* é realizada em duas etapas para cada objeto:

1. serão desenhados o *Cursor 3D* e o objeto, o *Depth Buffer* será configurado de modo a armazenar o menor valor de profundidade entre os dois. Caso a profundidade do *cursor* seja menor, o elemento não é selecionado e passamos para o próximo elemento da lista. Caso contrário, o cursor estará atrás da face frontal do objeto e precisamos verificar sua posição em relação a face posterior.
2. o *cursor* e o objeto são novamente rerenderizados, mas desta vez o *Depth Buffer* deverá guardar o maior valor de profundidade dentre os dois. Isso somente não é suficiente, devemos também descartar a face frontal do objeto, assim poderemos garantir que a comparação será feita com a face posterior.

Terminada a segunda fase do procedimento, o objeto estará selecionado caso a profundidade do *cursor 3D* seja menor que a da face posterior.

Em resumo, o algoritmo é simples, um objeto estará selecionado caso a coordenada z do *cursor* seja maior do que a da sua face frontal e menor do que a sua da posterior, fato que, combinado com as restrições anteriormente impostas pela seleção 2D, implica que o cursor encontra-se dentro do **Volume de Picking**.

Podemos observar que esta abordagem proposta para a seleção tridimensional tem um funcionamento correto apenas para objetos convexos, fato que motiva a pesquisa e desenvolvimento de um futuro e mais completo algoritmo.

Assim como na seleção bidimensional, o tratamento dos objetos selecionados deve ser definido pelo aplicativo, no entanto nenhum *dragger* será associado a eles.

7 Reestruturação do MTK

O MTK foi criado em 1995, e desde então trabalharam em seu desenvolvimento diferentes pessoas. Por este motivo, existia a necessidade de uma organização dessa biblioteca. Apesar da existência de uma filosofia e de padrões para confecção do seu código fonte, a estruturação do MTK encontrava-se ligeiramente inadequada ao suporte das novas funções a serem incluídas. Estas novas funções requerem métodos usualmente integrados às bibliotecas de sistemas de janelas (como os métodos tratadores de eventos) e às bibliotecas de modelagem geométrica (como os métodos geométricos para determinação de vetores normais).

Primando pela organização e legibilidade, realizamos um processo de reestruturação da biblioteca a fim de aperfeiçoá-la e fazer com que as modificações que venham a ocorrer futuramente tornem-se mais simples.

7.1 Concepção da Criação de uma Nova API

Inicialmente planejávamos a criação de uma nova classe com a finalidade de centralizar todos os recursos providos pelo MTK, ela seria a nova *API* da biblioteca.

A nova *API* deveria conter a antiga *API* (mtkCore), e os módulos responsáveis pela criação e manutenção da janela e dos dispositivos de entrada. Esta criação procurava trazer algumas vantagens para os usuários do MTK, a partir do momento em que não haveria mais a necessidade de alocação de diversas variáveis para a utilização das funcionalidades da biblioteca, e sim de apenas uma, já que os recursos estariam centralizados.

O problema é que esta alteração poderia, ao mesmo tempo em que facilita o desenvolvimento, limitar a flexibilidade permitida, visto que impediríamos o programador de alocar individualmente cada um dos recursos. Além disso, atentamos para o fato da modificação ser demorada e repleta de detalhes, o que tomaria tempo demasiado e fugiria do intuito do nosso projeto de pesquisa.

7.2 Reestruturação da API

Decidimos adequar a API da biblioteca a um padrão de projeto, visando tornar elegante a interação entre as classes e reutilizar soluções para problemas comuns em orientação a objeto.

Decidimos pela utilização do *Abstract Factory* [9].

A implementação de um *Abstract Factory* provê uma interface para a criação de famílias de objetos relacionados ou independentes sem a

necessidade da especificação de suas classes concretas. Este padrão de projeto consiste da definição dos tipos de classes: fábricas e produtos (concretos e abstratos). As classes concretas são geradas através de herança a partir das classes abstratas, que são instanciadas ao invés de seus equivalentes concretos. As fábricas concretas são classes cujos métodos tem a função de criar um produto concreto e instanciá-lo sobre a forma de um produto abstrato.

A reestruturação tornou mais simples a futura inclusão de funções e métodos referentes ao módulo do aplicativo que utilize outras bibliotecas gráficas diferentes do OpenGL, como por exemplo o DirectX. As novas classes para o módulo do aplicativo seriam incluídas como novos produtos concretos.

Para a adequação ao padrão, houve a criação de uma classe mtkCore, cuja função seria a de produto abstrato, enquanto a classe mtkCoreGL tornaria-se o único produto concreto.

Houve também a criação de classes de Fábrica. Foi criada uma fábrica abstrata a partir da qual seriam derivadas as concretas, uma para cada produto concreto a ser criado. No nosso caso foi criada apenas uma fábrica concreta, pois atualmente o MTK suporta apenas o OpenGL como biblioteca gráfica.

A fábrica concreta possui um método responsável pela criação do módulo do aplicativo. Surgiram dúvidas com relação a localização ideal para a chamada deste método. Optamos por deixar a chamada do método para o desenvolvedor, em detrimento da facilidade de programação, pois caso esta fosse feita dentro do código da biblioteca, seria necessário que o usuário possuísse todos os pacotes gráficos suportados pelo MTK instalados em seu computador. Atualmente, isto não seria um problema, pois como já dissemos, nossa biblioteca suporta apenas o OpenGL. Esta decisão foi tomada visando uma maior facilidade para os programadores no caso de uma futura expansão do MTK para novos pacotes gráficos.

Podemos observar a nova organização da API do MTK, o mtkCore da figura 1, na figura 2.

7.3 Reestruturação do Módulo de Seleção

O surgimento da idéia da utilização de mais de um dispositivo de entrada vem reforçar ainda mais a necessidade de uma reestruturação no módulo de seleção, que já se encontrava relativamente confuso.

Houve uma separação completa entre a seleção bidimensional e a tridimensional em duas classes mtkSelection2DGL e mtkSelection3DGL. As suas características comuns, como por exemplo o conceito de modos de seleção, foram mantidas em uma classe-pai (mtkSelection) e foram

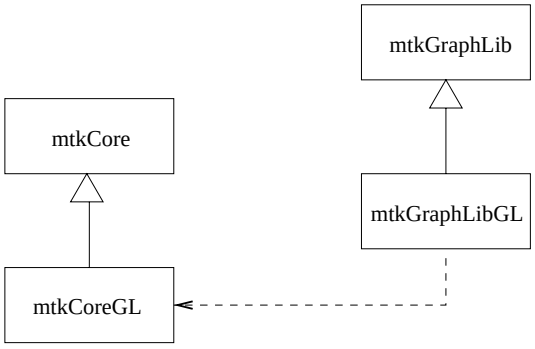


Figure 2: Nova organização da API do MTK.

atribuídas às duas subclasses através de um proceso de herança.

A seleção bidimensional continua praticamente sem grandes alterações, Já a parte referente a seleção tridimensional teve adicionada novas funcionalidades, como seu próprio método para seleção bidimensional (necessário devido a concepção desenvolvida para seleção 3D, da sua subdivisão em duas etapas) desconsiderando o modo de identificação de elementos.

Algumas modificações descritas em [11] permitem a utilização de mais de um dispositivo de entrada concomitantemente, mais especificamente é suportado o uso de dois dispositivos simultâneos, um bidimensional e outro tridimensional. Este fato acrescido da reestruturação do módulo de seleção permite que o MTK provenha seleções bidimensionais e tridimensionais tanto separada quando conjuntamente.

Surgiram dois grandes problemas:

- Impossibilidade do encapsulamento do método responsável pela seleção tridimensional [10] na classe mtkSelection3D

  Este problema aconteceu devido ao algoritmo proposto para resolução do problema da seleção 3D [10]. O algoritmo busca uma solução elegante e mais eficiente, através da utilização do conceito de volume de *picking* e do redesenho dos elementos da cena um a um. Para o redesenho dos elementos, o método precisa fazer acessos à classe mtkDisplayList, algo que não é possível de dentro do módulo de seleção, devido ao conceito do encapsulamento implementado no MTK.

  Buscamos uma solução deste problema dividindo a função pick3D em duas partes: a primeira é responsável pela construção da lista de elementos selecionados e utiliza o redesenho de objetos e a segunda consiste no tratamento desta lista de acordo com o modo de seleção. Após esta divisão, a segunda parte seria incluída na classe mtkSelection3D fazendo com que apenas o imprescindível,

ou seja, a parte que necessita do acesso à classe mtkDisplayList esteja localizada fora do modulo de seleção.

- Necessidade de organização de uma estrutura que permita a utilização simultânea dos métodos de seleção bidimensional e tridimensional e que compartilhem um único conjunto de elementos selecionados

  A idéia da utilização de mais de um dispositivo de entrada, estende-se até o módulo de seleção, pois parece lógico que cada dispositivo utilizado tenha suas funcionalidades próprias, entre as quais enquadra-se a seleção de objetos.

  Duas soluções foram propostas para a resolução deste problema de *SelectionSet* único:

  – Na primeira, teríamos agregado à classe mtkCore apenas um atributo responsável pela seleção. Este atributo seria abstrato, ou seja, ele poderia assumir as formas de seleção 2D e 3D.
  Esta alternativa, no entanto, apresentou alguns problemas. Para que fosse agregado à classe mtkCore um atributo abstrato, a classe mtkSelection deveria possuir os métodos implementados nas suas duas classes-filho mtkSelection2D e mtkSelection3D, o que obrigaria a definição de diversos métodos individualmente desnecessários para cada classe. Outro problema encontrado foi o chaveamento das seleções bidimensional e tridimensional, com relação ao local em que seria feito. Não havia um evento bem definido onde ele pudesse ser realizado. E, por último, o problema mais grave, o *SelectionSet* era perdido durante o chaveamento entre as seleções.

  – Na segunda solução optamos por agregar dois atributos de seleção à classe mtkCore, um para seleção 2D e outro para a 3D. Resolvemos separar o *SelectionSet* das classes responsáveis pela seleção 2D e 3D (mtkSelection2D e mtkSelection3D), criando uma nova classe para abrigá-lo (mtkSelectionSet). Definimos então o *SelectionSet* como um atributo protegido e estático e agregamos a classe *mtkSelectionSet* às duas classes responsáveis pela seleção. O fato do atributo ser definido estaticamente faz com que ele seja o mesmo para todas as instâncias da classe, resolvendo o problema do *SelectionSet* único.
  Esta solução permite ainda uma maior flexibilidade, pois o programador da biblioteca pode definir modos de seleção diferentes para cada uma das seleções.

Podemos observar a nova estrutura da classe mtkSelection da figura 1

na figura 3.

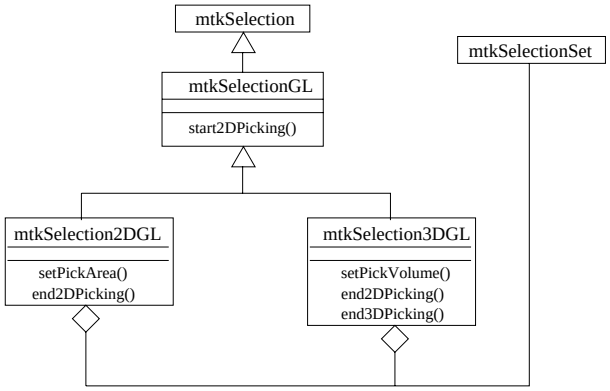


Figure 3: Nova organização da classe de seleção do MTK.

8 Movimento Restrito a Superfícies

Buscando um maior suporte na implementação de interfaces interativas 3D com mais precisão, decidimos disponibilizar aos usuários da nossa biblioteca uma nova opção de navegação com o *Cursor 3D*, na qual o movimento deste está restrito (*snapped*) à geometria de uma superfície arbitrária definida por um aplicativo.

8.1 Projeto Conceitual

Para o desenvolvimento deste tipo de movimento utilizamo-nos do conceito do plano tangente, um conceito comum a todas as superfícies suaves. Nosso algoritmo consiste em mapear o movimento realizado no plano XY pelo *mouse* para o plano tangente à superfície no ponto em que está localizado o *cursor* e, depois disso, projetar na superfície ao longo do normal do plano tangente o ponto obtido. Após a projeção deste ponto, um novo ponto sobre a superfície é obtido e o processo é repetido, podemos observar este processo na figura 4.

No nosso algoritmo praticamente todo o trabalho é realizado pelo aplicativo, visto que apenas este e não o MTK tem conhecimento sobre a geometria da superfície especificada. O aplicativo deve obter a projeção do ponto na superfície em consideração, fornecer as direções de deslocamento $\vec{d}$ do *cursor* no plano tangente correspondente. Podemos observar o esquema da figura 5.

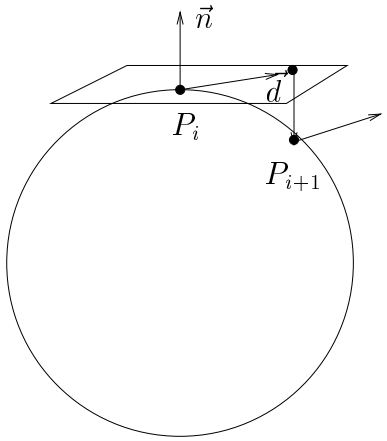


Figure 4: Modelo de movimento restrito à geometria.

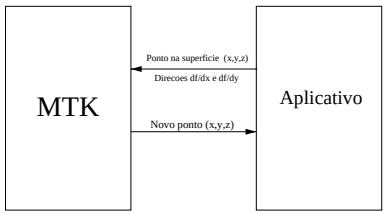


Figure 5: Esquema do projeto do cursor integrado

8.2 Problemas

8.2.1 Vetor Normal à Superfície

Inicialmente gostaríamos que a própria biblioteca pudesse descobrir o valor do vetor normal à superfície.

Pensamos em definir a normal à superfície através do cálculo do vetor gradiente, no entanto isso não foi possível, primeiro devido ao fato que o MTK não conhecia a geometria da superfície definida e depois porque a superfície não precisa ser necessáriamente definida por uma função implícita.

Surgiu a idéia de que, se durante o processo de renderização suportado pelo OpenGL as normais nos pontos da superfície são calculadas interativamente, podemos de alguma maneira descobrir seus valores e fazer uso deles no processo de interação.

Em Computação Gráfica, distinguem-se três modelos de tonalização:

- Tonalização Constante: a cor determinada por um modelo de iluminação (por exemplo, o modelo de Phong, traçado de raio

ou até radiosidade) em um ponto da face é atribuída a todos os outros pontos da face. Ou seja, considera-se neste caso que as faces são planas e que a direção dos raios luminosos é constante em relação a cada uma delas.

- Tonalização de Gouraud: utiliza-se o modelo de iluminação para calcular a cor em alguns pontos mais representativos da face e aplica-se uma interpolação bilinear em cima destas cores para obter as intensidades em outros pontos da face. Usualmente, escolhe-se como pontos representativos os vértices das faces.
- Tonalização de Phong: são determinados os vetores normais dos pontos representativos, como na tonalização de Gouraud, e a interpolação bilinear é aplicada em relação a estes vetores para obter as normais nos outros pontos. O modelo de iluminação é, então, aplicado diretamente em todos os pontos de interesse utilizando os vetores normais interpolados ou exatos para obter os valores de intensidade luminosa.

Infelizmente a biblioteca gráfica utilizada na versão corrente do MTK, o OpenGL, provém em sua implementação apenas duas opções para modelo de tonalização, FLAT e SMOOTH. Estas opções equivalem respectivamente a tonalização constante e de Gouraud (*Constant* e *Gourand Shading*). Temos somente acessos às normais dos vértices das faces através da opção GL_CURRENT_NORMAL.

8.2.2 Fornecimento apenas do Vetor Normal

No primeiro esboço de nosso algoritmo, o aplicativo deveria fornecer para o MTK não as direções, mas apenas o vetor normal à superfície no ponto, já que como vimos acima, não seria possível que o MTK realizasse o cálculo deste.

Com o vetor normal poderíamos então encontrar o plano tangente à superfície no ponto onde deveria ser realizado o movimento. Apesar disso, apenas o vetor normal ao plano tangente não era suficiente, precisávamos, no mínimo de mais um vetor para orientação. Com apenas um vetor não se podia determinar a direção do deslocamento que deveria ser mapeado do plano XZ para o plano tangente.

Para resolvermos este problema definimos que o aplicativo deveria fornecer dois vetores que identificassem o plano tangente. Escolhemos os dois vetores, ortogonais entre si, que definem as direções X e Y neste plano.

A escolha dessa opção se deve ao fato do mapeamento do deslocamento tornar-se mais fácil. Como os dois vetores estão contidos no plano tangente e são ortogonais, qualquer deslocamento neste plano

pode ser expresso através de uma combinação linear destes dois vetores.

Caso o vetor normal fosse escolhido em conjunto com uma destas direções, deveria ser aplicado um conjunto de transformações afins. Deveríamos aplicar uma translação seguida de rotação com intuito de transformar o plano tangente no plano xz, aplicar o deslocamento, e logo em seguida, aplicar a inversa destas transformações para retornarmos ao sistema inicial.

A combinação linear dos vetores utiliza-se de multiplicações entre escalares e vetores de ordem três, enquanto a composição de transformações é obtida através da multiplicação de matrizes quadradas de ordem quatro. Como o cálculo do deslocamento no plano tangente é realizado a todo momento, a opção pelos dois vetores contidos no plano apresenta vantagem, pois o processamento necessário neste caso seria menor, tornando o aplicativo mais veloz.

8.2.3 Implementação

A implementação do movimento do cursor restrito a superfícies é baseada em funções de *callback*. Uma *callback* é uma função registrada pelo programador que é disparada na ocorrência de algum evento definido.

O aplicativo deve registrar três funções de *callback* para a superfície: duas para o cálculo dos vetores que determinam a direção do plano tangente, definidas em setElementConstrainedDirectionX e Y e outra para a projeção do ponto na superfície, em setElementConstrainedProj. Estas funções são disparadas no movimento do *mouse* sobre a janela. Quando este evento ocorre, as direções são calculadas, via *callback*, e o deslocamento é mapeado, pelo MTK, no plano tangente. O ponto resultante deste mapeamento é então projetado na superfície, novamente através de uma função de *callback*.

As superfícies sobre as quais se deseja restringir o movimento devem ser definidas normalmente, como elementos da biblioteca. Desta maneira, cada superfície é associada a um identificador unívoco e pode ter suas próprias funções de *callback* registradas. Com isso, qualquer elemento definido pelo usuário pode ter o movimento do cursor restrito a sua geometria.

A escolha do tipo de movimento é feita através de uma função da biblioteca, mtkxsetMotionType. Ela seleciona um dos três tipos de movimento para o *Cursor3D* providos pelo MTK:

- MTKX_MAPPING_PLANES: Movimento livre pelo espaço
- MTKX_MAPPING_IMPLICIT: Movimento restrito a superfícies definidas por funções implícitas

- MTKX_MAPPING_CONSTRAINT: Movimento restrito a superfícies arbitrárias definidas pelo aplicativo.

8.2.4 Representação Gráfica do Cursor 3D

Devido ao acréscimo de um novo tipo de movimento aos já suportados pelo MTK, precisávamos de um modo simples para diferenciá-lo dos demais. Surgiu então a idéia de uma nova representação para o cursor 3D, pois em nossa concepção tratava-se de uma maneira intuitiva para a identificação do tipo de movimento corrente pelo usuário da biblioteca.

Definimos então uma nova representação gráfica para o *Cursor3D* de modo a caracterizar o movimento restrito às superfícies. Optamos pela mudança da cor padrão do cursor de branca para amarela e escolhemos também uma nova representação para seus eixos, de forma que o eixo y do cursor esteja sempre na mesma direção do vetor normal à superfície.

O módulo responsável pelo desenho do *Cursor3D* foi reformulado, incluímos uma nova representação para o cursor quando o movimento deste estiver restrito às superfícies, além de novas opções para que o programador possa escolher sua própria representação para o cursor.

Para a implementação desta funcionalidade foi adicionado um atributo à classe mtkCursor de um tipo enumerado onde é definido o estilo do cursor 3D. Ele pode assumir os seguintes valores: MTK_CURSOR_STYLE_NORMAL, MTK_CURSOR_STYLE_CONTRAINED, MTK_CURSOR_STYLE_OWN.

Os dois primeiros valores são utilizados para o movimento normal e restrito às superfícies do cursor 3D, respectivamente, e possuem uma representação gráfica já definida dentro da biblioteca. Caso o atributo assuma o terceiro valor, o programador deve registrar uma função de *callback*, reponsável pelo desenho da representação gráfica desejada para o cursor.

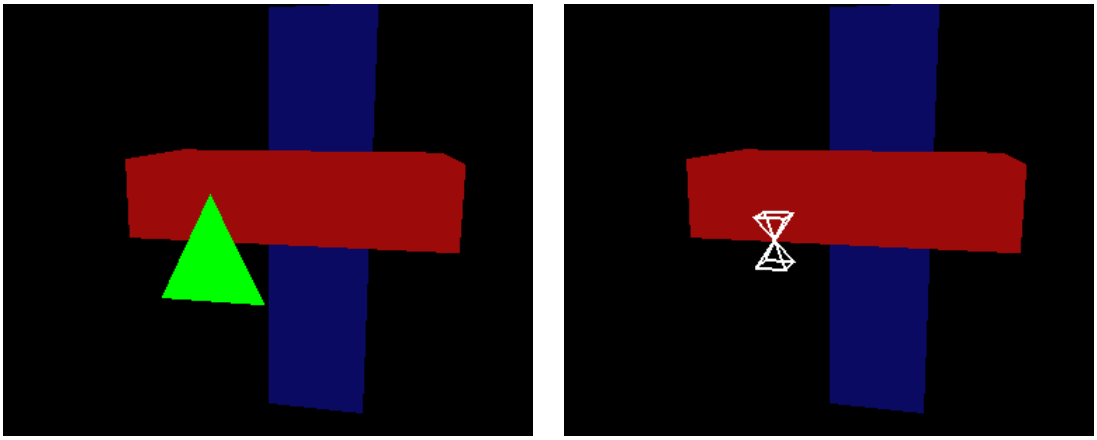
Podemos observar na figura 6 algumas representações alternativas para o *Cursor3D* na nova versão do MTK.

8.2.5 Experimentos

Foram desenvolvidos dois aplicativos onde o movimento do *Cursor3D* está restrito. Tais explicativos foram testados com dois tipos de dispositivos de entrada, *mouse* convencional e *spaceball* [11].

- Esfera

  No primeiro deles o movimento é realizado sobre uma esfera. Podemos observar as imagens capturadas do aplicativo na figura 7.



(a) (b)

Figure 6: Exemplos de representações gráficas para o Cursor3D

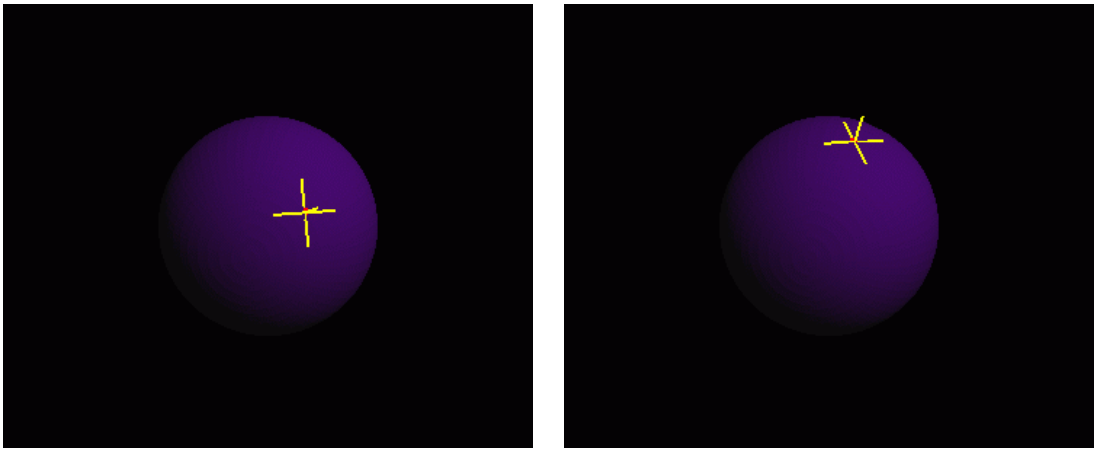
Segue um trecho do código, onde são registradas as funções de *callback* responsáveis pelo cálculo da direção do movimento do cursor no plano tangente e pela projeção dos pontos na superfície da esfera.

```
void func (gmPoint3 &P)
{
  gmVector3 V;

  // Vetor do centro ao ponto P
  V[0] = P[0];   V[1] = P[1];  V[2] = P[2];
  V = V.normalize();
  // Novo ponto recebe o valor do vetor normalizado multiplicado pelo raio
  P[0] = 4*V[0]; P[1] = 4*V[1]; P[2] = 4*V[2];
}


// retorna o vetor normal ao plano do grande circulo
void derfuncx (gmPoint3 p, gmVector3 &V)
{
  gmVector3 P, N;

  // vetor do centro ao ponto P
  P = p - gmPoint3(0,0,0);
  // vetor do centro ao polo norte da esfera
  N = gmPoint3(0,4,0) - gmPoint3(0,0,0);
```



(a) (b)

Figure 7: Movimento restrito a superfície da esfera

```
  V = gmCross(N,P);
  V = V.normalize();
}

// retorna produto vetorial entre o vetor normal ao plano
//do grande circulo e o vetor gradiente
void derfuncy (gmPoint3 p, gmVector3 &V)
{
  gmVector3 P, X;

  // calculo dos vetores dx e normal
  derfuncx(p, X);
  P = p - gmPoint3(0,0,0);
  // calculo do vetor dy
  if (p[2] > 0)
    V = gmCross(P,X);
  else
    V = gmCross(X,P);
  V = V.normalize();
}

.
.
.

// registro das callbacks
```

```
core->setElementConstrainedDirectionX(1, derfuncx);
core->setElementConstrainedDirectionY(1, derfuncy);
core->setElementConstrainedProj(1, func);
```

Para o desenvolvimento do aplicativo as direções de deslocamento sobre a esfera foram obtidas baseadas no trabalho de Clifford [8].

- Cubo

  O segundo aplicativo foi realizado sobre um cubo centrado na origem. As imagens capturadas podem ser vistas na figura 8.

(a) (b)

Figure 8: Movimento restrito a superfície do cubo

Neste exemplo tanto o fornecimento das direções quanto a realização das projeções ocorrem sobre as faces do cubo. Quando o movimento ultrapassa os limites definidos pelas arestas do cubo em algum dos eixos, é feito um cálculo da magnitude deste deslocamento fora do objeto, através da diferença entre as coordenadas do ponto (obtido após o deslocamento) e da aresta. Este valor é acrescido a uma das outras coordenadas do ponto e uma consequente projeção é realizada sobre a face adjacente.

9 Documentação e Web Site

Após a reestruturação do código fonte do MTK, decidimos também, começar o projeto de um novo site para a biblioteca.

Esta tarefa foi realizada em conjunto com meu colega de trabalho, Marcel Abrantes, responsável principalmente pela parte gráfica do web

site. Buscávamos um *lay-out* simples e ao mesmo tempo apresentável como o de uma biblioteca gráfica 3D.

Houve também necessidade de iniciar o desenvolvimento de uma documentação adequada para as funcionalidades providas pela API da biblioteca, cuja finalidade é facilitar sua utilização por parte dos programadores. Entretanto, trata-se de uma tarefa demasiadamente longa e um pouco complicada, devido às constantes alterações aos módulos da biblioteca durante o projeto. Até o momento, foram documentados apenas os métodos referentes aos módulos de seleção e do *display list*.

A idéia tem como intuito disponibilizar a biblioteca e sua documentação, assim como exemplos de utilização, para que programadores possam usufruir das funcionalidades providas pelo MTK.

10 Análise e Conclusões

Após o período trancorrido de trabalho pudemos concluir praticamente todas os itens descritos na seção 3.

As tarefas denotadas pelos itens 1, 2 e 3 foram terminadas e seus resultados são apresentados nas seções anteriores deste trabalho.

A comparação do desempenho da movimentação do *Cursor 3D* por dispositivos de entrada bidimensionais e tridimensionais (item 4) ainda não foi realizada.

A elaboração da documentação da API da biblioteca (item 5) não foi totalmente finalizada, pelo fato de ser um trabalho muito extenso e complicado devido às constantes alterações sofridas pela biblioteca durante o projeto.

A análise de todo o trabalho desenvolvido nos permite concluir que os objetivos almejados foram de certa forma alcançados, o *Cursor 3D* atualmente possui novas e interessantes funcionalidades que conseguiram ampliar sua interação com os objetos da cena tridimensional. Ainda foram realizadas tarefas como o início da documentação da API do MTK e a reestruturação desta biblioteca, que visam facilitar futuras modificações por parte dos desenvolvedores e tornar o uso da biblioteca mais fácil para os programadores.

Durante o período de desenvolvimento do trabalho, decidimos pela confecção de um artigo; em co-autoria com Wu, Abrantes e Batagelo; submetido como contribuição para o Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (Sibgrapi). Existe ainda a intenção da inclusão deste trabalho no Workshop de IC do Sibgrapi 2003.

11 Proposta para Trabalho Futuro

Podemos enumerar alguns dos seguintes itens para serem realizados em um trabalho futuro:

1. Término das tarefas restantes
   (a) Documentação da biblioteca, reestruturação do código e confecção do web site
   (b) Comparativo entre interatividade do Cursor3D e do Spaceball
2. Estudo sobre sistemas de janelas e conceito de sinais
3. Instanciação do MTK para outros sistemas de janelas
4. Expansão do algoritmo de seleção tridimensional para identificação correta de objetos côncavos
5. Estudo sobre funcionalidades e programação em placas gráficas
6. Alteração sobre o movimento restrito a superfícies, visando menor dependência do aplicativo e maior utilização dos recursos de hardware (aquisição de uma placa gráfica de última geração)
7. Depuração e testes

Apresentamos abaixo o novo cronograma como sugestão para continuidade do projeto.

Ativ.	Mai–Jun	Jul–Ago	Set–Out	Nov–Dez	Jan–Fev	Mar–Abr
1(a)	X	X	X	X	X	X
1(b)	X	X				
2	X					
3		X	X	X		
4	X	X				
5	X	X	X			
6			X	X	X	X
7	X	X	X	X	X	X

References

[1] Pierce, J.S., Forsberg, A.S., Conway, M.J., Hong, S. P., and Zeleznik, R.C. "Image Plane Interaction Techniques in 3D Immersive Environments", Proceedings of 1997 Symposium on Interactive 3D Graphics, (Providence, Rhode Island, April 27-30, 1997).

[2] Conner, D.B., Snibbe, S.S., Herndon, K.P., Robbins, D.C., Zeleznik, R.C. and van Dam, A. "Three-Dimensional Widgets", Computer Graphics (Proceedings of the 1992 Symposium on Interactive 3D Graphics), 25(2), ACM SIGGRAPH, March, 1992, pp. 183-188.

[3] Gleem - OpenGL Extremely Easy to use Manipulators
`http://www.media.mit.edu/~kbrussel/gleem/`

[4] Minority Report - Twentieth Century Fox and DreamWorks Pictures film
`http://www.minorityreport.com/`

[5] The Source for Java (TM) Technology
`http://java.sun.com/`

[6] C++
`http://www.cplusplus.com/`

[7] Open Inventor - SGI object-oriented toolkit for developing interactive 3D graphics applications
`http://www.sgi.com/software/inventor`

[8] Clifford, A.S. "Getting Arround on a Sphere". Graphics Gems II, Academy Press - 1990

[9] Data & Object Factory - Developer Training Home Page
`http://www.dofactory.com/patterns/patterns.aspx`

[10] Tost, Daniel. Extensão das Funções de Emulação do Cursor 3D no MTK. *Relatório Parcial de Projeto de Pesquisa de Iniciação Científica FAPESP. Processo nº 02/01162-0.* Novembro 2002.

[11] Abrantes, Marcel. Desenvolvimento de uma Interface entre MTK e Spaceball. *Relatório Final de Projeto de Pesquisa de Iniciação Científica FAPESP. Processo nº 02/01161-3.* Maio 2003.

[12] MTK - Manipulation Tool Kit
`http://www.dca.fee.unicamp.br/~ra003168/mtk/`

[13] Interfaces hápticas
`http://haptic.mech.nwu.edu/`

[14] Bier, E. A., "Snap-Dragging In Three Dimensions," Proc. 1990 Symposium on Interactive 3D Graphics, Computer Graphics, 24 (2), pp. 193-204.