MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-12982-PRE/8259

RECONSTRUÇÃO GRÁFICA TRIDIMENSIONAL DE CONSTRUÇÕES A PARTIR DE IMAGENS AÉREAS

Daniel Moisés Gonzalez Clua

*Bolsista UNIVAP

Relatório Final de Projeto de Iniciação Científica (PIBIC/CNPq/INPE), orientado pelo Dr. Valdemir Carrara

INPE
São José dos Campos
2005

RECONSTRUÇÃO GRÁFICA TRIDIMENSIONAL DE CONSTRUÇÕES A PARTIR DE IMAGENS AÉREAS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/CNPq/INPE)

Daniel Moisés Gonzalez Clua (UNIVAP, Bolsista PIBIC/CNPq)
E-mail: dmgc29785@yahoo.com.br

Dr. Valdemir Carrara (DMC/ETE/INPE, Orientador)
E-mail: val@dem.inpe.br

Julho de 2005

SUMÁRIO

RESUMO

Este trabalho, iniciado em abril de 2005, tem como objetivo o desenvolvimento de algoritmos para a obtenção das dimensões de construções de edifícios a partir do processamento de imagens aéreas de alta resolução. Estas dimensões serão posteriormente utilizadas para compor objetos gráficos tridimensionais utilizando a própria textura obtida da imagem. No presente trabalho, os vértices das construções deverão ser fornecidos por um dispositivo apontador. O trabalho compreende também a eliminação da paralaxe, a correção da iluminação em função do ângulo de elevação solar, e a eliminação de sombra na textura. A metodologia a ser utilizada no desenvolvimento da presente proposta envolve a aplicação de geometria analítica e vetorial no desenvolvimento de algoritmos para compor as dimensões das construções tridimensionais. O algoritmo será desenvolvido em linguagem C ou C++, com visualização realizada por meio de OpenGl. Os principais planos para o trabalho envolvem: elaborar uma revisão bibliográfica a respeito da elaboração poligonal e geometria vetorial, familiarizar-se com técnicas de elaboração poligonal do tipo OpenGl e DirectX, desenvolver algoritmos para composição de objetos geométricos simples e para a manipulação destes em imagens, e para efetuar transformações e correções em imagens. Tais algoritmos serão implementados para compor objetos gráficos tridimensionais a partir das informações obtidas das imagens, com aplicação de texturas. Mais adiante, os códigos serão otimizados e novos recursos serão implementados.

ABSTRACT

The main objective of this work is to develop algorithms to reconstruct, in a 3D graphic environment, the urban buildings based on high-resolution aerial images taken from satellites or airborne cameras. The image will also serve as a graphical texture to compose the virtual environment. Building dimensions, mainly height, will be calculated based on points in the image, previously selected by a pointer device (mouse). The texture image will be processed in order to correct for intensity due to Sun's elevation angle, to eliminate or reduce the building shadow on ground, and to erase as much as possible the parallax, i. e., the projection of the building sides in the ground texture. The algorithm will be programmed in C or C++, and OpenGl will be employed to visualize the reconstructed buildings.

CAPÍTULO 1

INTRODUÇÃO

A construção de ambientes gráficos tridimensionais (ambiente virtual) no computador tem encontrado diversas aplicações recentemente. Embora ambientes virtuais sejam freqüentes em jogos, cenários tridimensionais que representam ambientes reais são cada vez mais empregados no turismo, no urbanismo, no tratamento de distúrbios nervosos (fobias) e simuladores de vôo, veículos e máquinas. Também é usual, em tais ambientes, a reconstrução virtual de partes de cidades ou mesmo cidades inteiras. Uma vez que esta reconstrução pode ser bastante complexa e difícil, em virtude dos vários cenários existentes, buscam-se ferramentas automáticas ou semi-automáticas para tal reconstrução (VTP 2005, Carmenta 2005). Imagens aéreas e imagens obtidas por satélites com alta resolução são utilizadas usualmente, o que permite reconstruir grandes áreas com poucas imagens. O processo de reconstrução virtual pode ser dividido em duas partes: na primeira buscam-se formas automáticas ou semi-automáticas de extração de características (ou informações) das imagens para detectar as construções, e na segunda efetua-se a reconstrução propriamente dita. Pretende-se neste trabalho investir na reconstrução de ambientes urbanos (edifícios e casas) a partir de informações previamente fornecidas a um programa computacional, o que caracteriza um método semi-automático.

Portanto, o objetivo deste trabalho é elaborar um programa que permite reconstruir espacialmente e visualizar construções e edifícios a partir de imagens aéreas. A linguagem usada é C++, com recursos de OpenGL para a visualização.

O OpenGL é definido como “uma interface em *software* para *hardware* gráfico”. Em essência, é uma biblioteca de gráficos tridimensionais e modelagem, portátil e rápida, o que permite gerar imagens interativas em tempo-real. O OpenGL não é uma linguagem, como C ou C++, mas sim uma biblioteca de funções pré-programadas. Na realidade não existe um “programa em OpenGL” mas sim um programa que o desenvolvedor escreve usando o OpenGL como um de seus APIs.

As principais funções a serem implementadas para efetuar a reconstrução virtual são:

1) Exibir uma imagem e permitir ao usuário o fornecimento de pontos (vértices) que definem a construção, que irá definir um prisma vertical com base não necessariamente retangular.
2) Obter parâmetros que permitam avaliar a altura da construção a partir da sombra projetada na imagem ou da própria altura projetada (em caso de vista em perfil).
3) Extrair a textura a ser aplicada à construção da própria imagem.
4) Eliminar ou minimizar a sombra do edifício na textura do solo.
5) Eliminar ou minimizar a imagem projetada do edifício na textura do solo.
6) Exibir o resultado na forma tridimensional.

1.1 Organização do documento

Este relatório parcial foi dividido em dois capítulos:

- CAPÍTULO 2 – DESENVOLVIMENTO – Aqui serão descritos os métodos e funções estudados e utilizados no trabalho até o momento.
- CAPÍTULO 3 – CONCLUSÕES - Conclusões parciais sobre os métodos estudados e analisados e próximas atividades.

CAPÍTULO 2

DESENVOLVIMENTO

Para cumprir os objetivos do trabalho e para implementar as funções descritas no capítulo precedente, procedeu-se a um estudo dividido em duas partes: na primeira formulam-se as relações geométricas que permitem efetuar a reconstrução a partir de informações colhidas das imagens, e, na segunda, implementam-se o equacionamento em OpenGL de forma a visualizar o resultado.

2.1 – Formulação geométrica

A formulação geométrica possui por objetivo determinar a altura dos edifícios a partir de informações da própria imagem. Tomando como exemplo a imagem mostrada na Figura 2.1, deseja-se obter a altura do edifício em unidades de pixel, representada pela magnitude do vetor *H* perpendicular ao plano da imagem, e cuja projeção no plano da imagem é o vetor *h*.

Nota-se que, embora a altura do edifício não seja vista na sua real dimensão na imagem, por outro lado a sombra tem sua verdadeira dimensão (ou bem próxima dela), se for suposto que a câmera tenha seu eixo perpendicular ao solo, como ilustra a Figura 2.2. Contudo, se o ângulo α que representa a elevação do Sol sobre o horizonte for conhecido, então a altura *H* do edifício pode ser obtida por meio do comprimento da sombra do edifício na imagem, ou seja, do módulo do vetor *s*, conforme mostra a figura 2.1, resultando então:

$$|H| = |s| \tan\alpha$$

Caso, porém, este ângulo não seja conhecido, pode-se ainda assim estimá-lo desde que se conheça a altura real de alguma construção e a verdadeira resolução da imagem, isto é, o tamanho de cada pixel, ou ainda o comprimento da sombra deste edifício. Seja então H'_m a altura em metros de uma determinada construção conhecida e

seja |*s*'| o comprimento de sua sombra na imagem (em pixels), também visto na Figura 2.1. Se *R* for a resolução (em metros/pixel), então a altura do edifício *H*' será dada por:

$$|H'| = \frac{H'_m}{R} \text{ (em pixels).}$$

Fig. 2.1 – Imagem aérea e os principais pontos que devem ser utilizados no processo de reconstrução. O contraste da imagem foi reduzido artificialmente para evidenciar os pontos a serem fornecidos externamente.

O ângulo de elevação do Sol pode agora ser obtido por:

$$\alpha = \arctan\left(\frac{|H'|}{|s'|}\right),$$

e poderá ser utilizado na reconstrução de outros edifícios da mesma imagem. Caso a resolução não seja conhecida, pode-se medir a distância real em linha reta horizontal entre dois pontos conhecidos de uma imagem e esta mesma distância na própria imagem. A relação entre as duas distâncias irá fornecer a resolução.

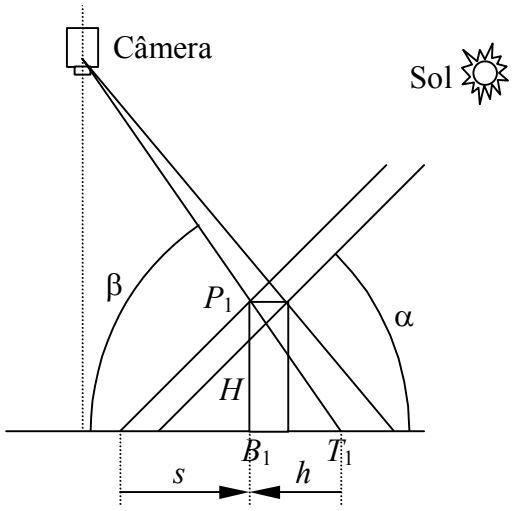


Fig. 2.2 – Vista lateral da geometria do problema.

Se o comprimento *s* da sombra não estiver disponível para um dado edifício (por exemplo, se a sombra for projetada sobre uma outra construção), pode-se estimar a altura tendo como base a altura projetada, desde que seja conhecida a altura *H'* e sua projeção *h'* de um outro edifício próximo. Neste caso emprega-se uma regra de proporcionalidade:

$$|H| = |h| \frac{|H'|}{|h'|}$$

É necessário que as duas construções estejam próximas, caso contrário os efeitos da projeção em perspectiva podem influir nos resultados, isto é, o ângulo β, mostrado na figura 2.2 deve ser aproximadamente igual em ambos os edifícios.

Sendo então fornecidos externamente os pontos T_1, T_2, T_3 e T_4, que delimitam o topo da construção, e o vetor *h* que representa a projeção da altura do edifício na imagem, as coordenadas dos pontos da base da construção, B_i, são facilmente obtidas

pela relação $B_i = T_i + h$. Estes pontos serão utilizados para a posicionar a base do edifício virtual. Num sistema de eixos cartesianos, onde x e y formam o plano da imagem e z é perpendicular a este plano, os pontos P_i do topo do edifício são obtidos por $P_i = B_i - H$.

2.2 – Visualização da reconstrução

O início do trabalho consistiu na familiarização dos principais comandos de programação em OpenGL. Estes envolviam a criação de uma janela simples, passando-se informações tais como a cor de fundo, o seu tamanho, sua escala, modo de visualização, etc. Os principais comandos OpenGl utilizados na visualização são comentados separadamente, em função do tipo de saída provocada. As informações coletadas para a configuração adequada do OpenGl e para a preparação do programa vieram do livro OpenGl Super Bible (Wright, 2000) e de programas tutores encontrados na Web (Neon-Helium Productions, 2005).

Para o correto funcionamento do OpenGl é necessário a inclusão no programa dos arquivos `gl.h`, `glu.h`, e `glaux.h` que contêm valores previamente associados a constantes. São também definidas nestes arquivos as especificações de tipos de variáveis, uma vez que o OpenGl, por ser multi-plataforma, estabelece regras para os tipos usuais, como `GLfloat`, `GLint`, etc. Estes arquivos em geral são fornecidos com o compilador C++, mas podem ser obtidos facilmente na *web* caso contrário. Além disso as seguintes bibliotecas devem ser incluídas no projeto para se poder utilizar os comandos de OpenGL no ambiente Windows: `OpenGL32.lib`, `GLu32.lib`, e `GLaux.lib`.

Os principais comandos do OpenGl utilizados na visualização do processo de reconstrução de edifícios são mostrados a seguir.

2.2.1 – Comandos para configuração da janela de visualização

Os comandos para configuração da janela de visualização permitem que se crie uma janela com dimensões previamente especificadas e o tipo de projeção a ser empregada na visualização.

- `gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f)`

Esta função especifica o tipo de projeção (perspectiva) e a pirâmide truncada de recorte. O primeiro parâmetro indica o ângulo (graus) de abertura na direção *y*. O segundo parâmetro fornece a razão de aspecto, isto é, a relação entre a altura e a largura da janela. Os demais indicam a profundidade mínima e máxima de recorte dos objetos.

- `glClearColor(0.0f, 0.0f, 0.0f, 0.0f);`

A função especifica a cor de fundo da janela em código RGB (Red-Green-Blue). No exemplo, a cor seria preta. O último parâmetro é a transparência, conhecido como canal alfa.

- `glShadeModel(GL_SMOOTH);`

A função glShadeModel indica o tipo de *shading*. O *smooth shading* usado no exemplo suaviza a transição de cores em um polígono, porque interpola as cores entre os vértices. A outra possibilidade é usar `GL_FLAT` que impede a interpolação.

- `glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);`

Esta função especifica aspectos que dependem da implementação do OpenGL utilizado. No caso, empregou-se correção de perspectiva que indica como deve ser feita a interpolação das texturas. Selecionou-se `GL_NICEST`, que oferece um melhor resultado, embora diminua um pouco a performance.

2.2.2 - Criação Objetos

A seguir, foram criados objetos por meio do comando `glVertex3f()`, onde os argumentos transferidos correspondem à posição de cada vértice do objeto na tela. Usando-se este comando, pode-se criar triângulos e quadriláteros e, unindo-se vários destes, obtém-se uma aproximação da forma tridimensional desejada. A sintaxe para a criação de um triângulo é:

- `glBegin(GL_TRIANGLES);`

Este comando informa que os próximos comandos irão fornecer os vértices de um ou mais triângulos. Para fornecer os vértices de quadriláteros deve-se utilizar `glBegin(GL_QUADS)`.

- `glVertex3f( 0.0f, 1.0f, 0.0f);`

Este comando define um vértice do objeto. Se o objeto a ser criado é um triângulo, este comando deve ser usado três vezes, um para cada vértice. Os três parâmetros correspondem às coordenadas *x*, *y* e *z* do vértice. Novos vértices podem seguir o primeiro triângulo, e a cada conjunto de 3 vértices um novo triângulo é formado.

- `glEnd();`

O `glEnd()` indica que todos os vértices do objeto já foram passados. Triângulos incompletos são descartados após `glEnd()`. Com o comando `glColor3f`, pode-se aplicar cores aos objetos:

- `glColor3f(1.0f,0.0f,0.0f);`

Os números passados como argumentos correspondem aos valores RGB da cor. Este comando deve ser acionado antes do fornecimento dos vértices e pode variar de vértice para vértice, como mostra a pirâmide da figura 2.3:

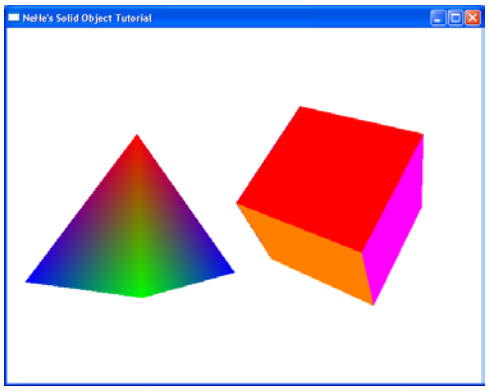


Fig. 2.3 – Pirâmide e Cubo com métodos de coloração diferentes.

2.2.3 - Movimento e Rotação

Objetos no OpenGl podem ser movidos e orientados por meio das funções `glTranslatef(x, y, z)` e `glRotatef (angle, x, y, z)`. As ações de movimentação devem ser efetuadas antes da criação dos objetos que serão movidos ou orientados pela ação. Estes comandos seguem a sintaxe dada abaixo:

- `glTranslatef(3.0f,0.0f,0.0f);`

Esta função indica a posição no espaço em que o objeto será desenhado (nos eixos *x*, *y* e *z*, respectivamente). Ao mudar-se estes valores de forma constante, o objeto desloca-se pela tela.

- `glRotatef(rquad,1.0f,0.0f,0.0f);`

A função `glRotatef` faz o objeto girar ao redor de um ou mais de seus eixos. O primeiro argumento corresponde ao ângulo de rotação. Para simular um movimento contínuo, este valor foi atribuído a uma variável que muda de forma constante. Os valores seguintes indicam o quanto o objeto irá girar em cada um dos eixos (*x*, *y* e *z*, respectivamente).

As funções `glTranslatef` e `glRotatef` atualizam o valor corrente da matriz de transformação, multiplicando-a pela matriz de translação, rotação ou variação de escala

desejada. Caso se deseje preservar uma dada matriz, como por exemplo numa árvore de articulações (um personagem com braços e pernas), deve-se utilizar as funções `glPushMatrix` e `glPopMatrix` descritas a seguir, para salvar e recuperar a matriz de transformação.

- `glPushMatrix();`
- `glPopMatrix();`

A função `glPushMatrix` salva a matriz corrente numa pilha (*stack*) do tipo LIFO (*last in, first out*, ou "o último a entrar é o primeiro a sair"). Simultaneamente, ela mantém a matriz corrente com seu último valor. Novas transformações podem ser aplicadas aos objetos que serão criados usando-se a translação e rotação já descritas aplicadas à matriz corrente. O valor anterior da matriz de transformação poderá ser então recuperado da pilha pela função `glPopMatrix`. A matriz corrente é inicializada como uma matriz identidade.

2.2.4 - Aplicação de Texturas

A aplicação de texturas permite melhorar o realismo do ambiente virtual pela aplicação de uma imagem à superfície de objetos. A primeira coisa a ser feita, ao se trabalhar com texturas, é criar um vetor de variáveis para a quantidade de texturas que serão utilizadas, como por exemplo `GLuint texture[1];`.

Para se aplicar uma textura a um objeto, deve-se primeiro carregar a imagem, criando a seguinte função:

- `AUX_RGBImageRec *LoadBMP(char *Filename){...}.`

Esta é uma função para fazer o armazenamento de um arquivo bitmap. A variável *Filename* é a que irá conter o nome do arquivo a ser carregado. Dentro desta função são colocados comandos para verificar se o arquivo realmente existe e, nesse caso retornar o arquivo, ou exibir uma mensagem de erro caso o arquivo não exista.

Numa outra função, que carrega a textura, deve-se declarar outro vetor, usando a função `AUX_RGBImageRec` já mencionada. Da mesma maneira, o vetor deve ter o número de posições correspondente à quantidade de texturas que serão utilizadas:

- `AUX_RGBImageRec *TextureImage[1];`

Entre outras informações, esta estrutura irá armazenar a altura (`sizeY`), largura (`sizeX`) e os dados (`data`) da imagem no formato bitmap.Em seguida, deve-se indicar o caminho da imagem a ser utilizada como textura e verificar se este caminho existe. Isto é feito através da função `LoadBMP()` criada anteriormente:

- `if (TextureImage[0]=LoadBMP("Data/imagem.bmp")){`

Foi usado um laço de condição do tipo `if` para verificar o caminho da imagem. Se estiver correto, ela será carregada e armazenada como textura, através dos comandos seguintes:

- `glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[0]->sizeX, TextureImage[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage[0]->data);`

O primeiro argumento indica que a textura será bidimensional (está-se usando a função `glTexImage2D`, portanto a textura deve ser bidimensional). O valor 0 se refere ao nível de detalhe. O nível 0 é o nível base da imagem e é o que se usa geralmente. Quanto maior o número, mais reduções *mipmap* serão feitas na imagem. O terceiro argumento indica o número de componentes de cor da textura e pode variar de 1 a 4 (vermelho, verde, azul e alfa). Os dois argumentos seguintes indicam os tamanhos vertical e horizontal da imagem (no exemplo, os tamanhos usados são os mesmos da própria imagem, obtidos através das informações contidas na variável `TextureImage[0]`). O valor seguinte é referente à borda, os valores possíveis são 0 e 1. O "`GL_RGB`" diz ao OpenGL que os dados da imagem usada são compostas de vermelho, verde e azul, nessa ordem. Existem outros oito tipos de valores nas bibliotecas do OpenGL que podem ser utilizados, cada um com um formato de cor diferente. O

argumento seguinte especifica os tipos de dados dos pixels da textura, sendo que existem oito tipos no OpenGL. O `GL_UNSIGNED_BYTE`, usado no exemplo, significa que os dados que criam a imagem são compostos por um byte sem sinal. E por último deve ser indicado um ponteiro para os dados da imagem. Novamente, isto foi feito através do `TextureImage[0]`, que contém estas informações.

- ```
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  ```

Estas duas linhas dizem ao OpenGL que tipo de filtro usar quando a imagem está maior (`TEXTURE_MAG`) e quando ela está menor (`TEXTURE_MIN`) do que seu tamanho original. O tipo de filtro usado no exemplo (`GL_LINEAR`) faz uma boa suavização na textura do objeto, tanto quando ele está longe como quando ele está perto, mas necessita mais processamento. Outros tipos de filtro possíveis seriam o `GL_NEAREST`, que oferece menos filtragem mas uma melhor performance e o `GL_LINEAR_MIPMAP_NEAREST`, que oferece o melhor resultado mas também requer o maior processamento entre todos.

Finalmente, deve-se aplicar a textura ao objeto a ser criado. Para isso, usa-se o comando, antes de criar-se o objeto, indicando a variável onde está armazenada a textura:

- ```
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  ```

Deve-se, também, relacionar cada vértice do objeto com um ponto da imagem da textura, através do comando `glTexCoord2f(x, y)`:

- ```
  glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
  ```

O ponto da imagem usado no exemplo seria o canto inferior esquerdo, pois o sistema de coordenadas usado pelas texturas varia de 0 a 1 tanto no eixo *x* (no sentido esquerdo para o direito) como no eixo *y* (no sentido de baixo para cima). Na figura 2.4 tem-se um exemplo de uma textura aplicada às seis faces de um cubo.
  ```

Fig. 2.4 – Aplicação de textura.

2.2.5 - Iluminação

A iluminação em uma cena virtual realça o aspecto tridimensional dos objetos, por meio de diferenças nas intensidades das cores entre as partes mais e iluminadas e menos iluminadas.

Há dois tipos importantes de iluminação que podem ser usadas em uma cena. A primeira é a iluminação ambiente que não vem de nenhum local em particular e ilumina todos os objetos da cena igualmente. A segunda é a iluminação direta, que vem de uma fonte num ponto específico da cena. Objetos atingidos pela luz ficarão bem iluminados enquanto as partes não iluminadas ficam escuras. Para ambos os casos, deve-se primeiro criar um vetor onde serão armazenados os valores que indicam a intensidade e a cor da luz:

- `GLfloat LightAmbient[]= { 0.5f, 0.5f, 0.5f, 1.0f };`

ou

- `GLfloat LightDiffuse[]= { 1.0f, 1.0f, 1.0f, 1.0f };`

Os três primeiros argumentos indicam a quantidade de cada cor que a luz emite. O quarto é o valor do canal alfa da luz, que especifica a intensidade e transparência da

luz. Luzes não-transparentes (alfa vale 1.0) tem a intensidade máxima dada por seus componentes de cor.

Como a luz direta provém de um ponto específico, deve-se indicar também a sua posição, através de outro vetor de variáveis:

- `GLfloat LightPosition[]= { 0.0f, 0.0f, 2.0f, 1.0f };`

Se o quarto argumento não for nulo, a luz é posicional e os três argumentos anteriores indicam sua posição nos eixos *x*, *y* e *z,* respectivamente. Se for nulo, a luz é direcional e os três argumentos anteriores indicam a direção em que a luz aponta.

Para gerar a iluminação, são usados os seguintes comandos (lembrando que os valores estão sendo passados pelos vetores criados anteriormente):

- `glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);`

que indica a cor e intensidade da luz ambiente da luz número um,

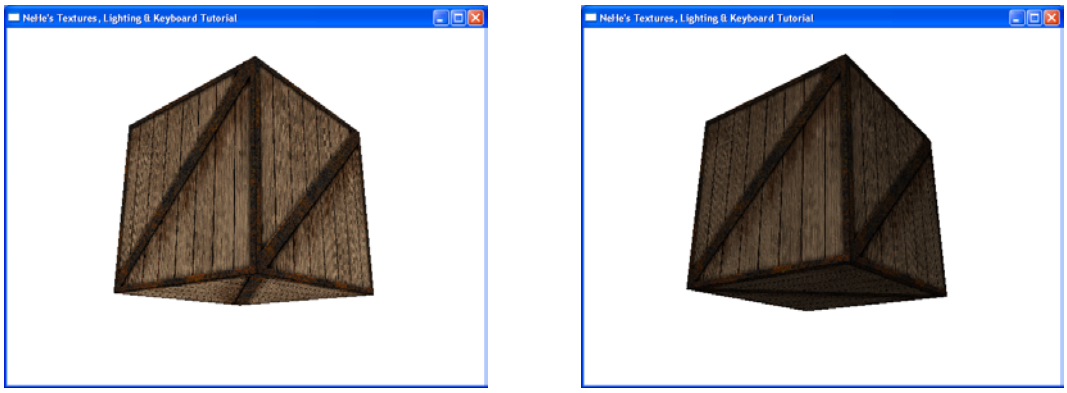
- `glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);`

que indica a quantidade de luz difusa, e

- `glLightfv(GL_LIGHT1, GL_POSITION, LightPosition);`

que indica a posição desta luz. Até oito luzes podem ser empregadas simultaneamente.

Para ativar e desativar a luz direta na cena, usa-se os comandos `glEnable(GL_LIGHTING)` e `glDisable(GL_LIGHTING)`, respectivamente. Se a luz está ativada, o programa usa os valores da iluminação difusa e ambiente. Caso contrário usa apenas os valores da iluminação ambiente. A figura 2.5 mostra a diferença entre iluminação ambiente (*a*) e iluminação difusa (*b*).



(*a*) (*b*)

Fig. 2.5 - Iluminação

2.2.6 – Entrada de Dados

Uma vez que os pontos utilizados para a reconstrução devem ser fornecidos externamente, então é necessário contar-se com recursos que permitam ao programa receber entradas do usuário, tanto pelo teclado como pelo mouse. Para identificar as entradas do teclado, primeiro deve-se criar um vetor de 256 variáveis booleanas, que indicarão a existência de uma determinada teclas sendo pressionada:

- `bool keys[256];`

Pode-se usar, então, um laço de condição do tipo `if` para indicar qual tecla está sendo pressionada, por exemplo:

- `if (keys['F']){...}`

Devido à letra estar entre aspas simples, é passado o valor numérico (ASCII) da letra no teclado e se verifica, então, se a tecla "F" está sendo pressionada.

Para as leituras de mouse, são usados os comandos seguintes para armazenar a posição do mouse, em pixels, quando o botão é clicado:

- `x = LOWORD(lParam);`

Com este comando, a variável `x` recebe o valor em pixels referente à distância do ponteiro do mouse à borda esquerda da janela.

- `y = HIWORD(lParam);`

Nesta função, a variável `y` recebe o valor em pixels referente à distância do ponteiro do mouse à borda superior da janela.

2.2.7 – Leitura de Parâmetros

O objetivo da leitura de parâmetros é permitir que o usuário, através do mouse, indique os pontos para a reconstrução de edifícios, a partir da exibição de uma imagem aérea na tela. Para tanto a imagem é aplicada como textura – usando os métodos descritos anteriormente – a um objeto retangular de mesmas proporções em vista frontal. O programa permite também que o usuário mova, acerque ou distancie a imagem, através de entradas do teclado.

Na implementação atual, o usuário deve fornecer 5 pontos: quatro para o topo do edifício (pontos T_1 a T_4, conforme a figura 2.1) e um para a base (ponto B_1). Uma vez que a leitura das coordenadas destes pontos, efetuada pelo mouse, estão em unidades de pixel da janela, é então necessário convertê-las para unidades de pixel da imagem por meio de uma transformação linear:

$$T_{xi} = a\ x_i + d_x$$
$$T_{yi} = a\ y_i + d_y$$

onde a é a ampliação (fator de escala) utilizada na visualização. Estes valores são então atribuídos a dois vetores, um para as coordenadas no eixo x e outra para as do eixo y. Ambos possuem cinco valores: quatro para os vértices do topo do edifício e um para um vértice da base do edifício que deve corresponder ao primeiro vértice do topo. Este quinto valor serve para calcular o deslocamento que o programa deve aplicar ao objeto para que ele se origine da base do edifício na imagem e não do topo. Passados estes pontos, o programa constrói o objeto que representa o edifício, usando os valores

armazenados das coordenadas, como visto nas figuras 2.6 e 2.7. Na versão mostrada aqui, a altura do edifício é fixa, ou seja, não foi utilizado um cálculo para a determinação da altura.

Fig. 2.6 – Pontos recebidos para a construção do edifício.

Fig. 2.7 – Edifício reconstruído.

CAPÍTULO 3

CONCLUSÕES

Nesta fase inicial de desenvolvimento considera-se que os objetivos, que consistiam de familiarização com as funções do OpenGL, foram atingidos. Os próximos aprimoramentos do trabalho podem ser resumidos em:

- Aplicação de textura ao edifício,
- Implementação de métodos de cálculo da altura,
- Algoritmos para construir um edifício com diversas formas, não apenas retangular,
- Demais funções listadas no Capítulo 1.

REFERÊNCIAS BIBLIOGRÁFICAS

Wright, S. R.; Sweet, M. *OpenGL Super Bible. Second Edition*. Waite Group Press, 2000.

Neon-Helium Productions, *NeHe*, 2005 (http://nehe.gamedev.net/)

VTP - *The Virtual Terrain Project*, 2005 (http://www.vterrain.org/)

Carmenta – *SpatialAce*, 2005. (http://www.spatialace.com/)