Geração de imagens com texturas utilizando OpenGL

CRISTIANO CACHAPUZ E LIMA

Universidade da Região da Campanha
CCEI - Centro de Ciências da Economia e Informática
Av. Tupy Silveira, 2099, 96400-030 Bagé, RS, Brasil
e-mail: ccl@urcamp.tche.br
Homepage: http://www.urcamp.tche.br/~ccl/

Resumo. Esta pesquisa em forma de artigo discute algumas técnicas necessárias à aplicação de imagens de textura sobre polígonos utilizando-se a interface de programação de aplicações gráficas OpenGL, para a obtenção de realismo na geração de imagens via computador.

Abstract. This paper discusses some techniques to bind texture images on polygons using the OpenGL API, in order to obtain realism in computer generated images.

Keywords: Computer Graphics, Image Processing, Texture, OpenGL.

1. Introdução

OpenGL (ou "Biblioteca Gráfica Aberta") é uma interface de software para hardware gráfico e que consiste de um conjunto de centenas de procedimentos e funções que permitem a um programador especificar os objetos e operações envolvidas na produção de imagens gráficas de alta qualidade, especificamente imagens coloridas de objetos tridimensionais. [Segal--Akeley (1997)]

A OpenGL é estritamente definida como “um software de interface de hardwares gráficos”. Essencialmente, é uma biblioteca para gráficos e modelagem 3D que é extremamente portável e muito rápida. Usando-se OpenGL, pode-se criar gráficos 3D bonitos e elegantes com qualidade muito próxima de um ray-tracer. A maior vantagem de se usar OpenGL é que ela é muitíssimo mais rápida que um ray-tracer. Ela usa algoritmos cuidadosamente desenvolvidos e otimizados pela Silicon Graphics, Inc. (SGI), reconhecidamente empresa líder mundial em computação gráfica e animação. [Wright Jr--Sweet (1996)].

Para o programador, a OpenGL é um conjunto de comandos que permitem a especificação de objetos geométricos em duas ou três dimensões, junto com comandos que controlam como estes objetos são renderizados no buffer de imagem. [Segal--Akeley (1997)].

Um programa típico que usa OpenGL começa com chamadas para abrir uma janela no framebuffer no qual o programa desenhará. Então, são feitas chamadas para alocar um contexto GL e associá-lo com a janela. Uma vez um contexto de GL ser alocado, o programador está livre para escrever comandos em OpenGL. Algumas chamadas são usadas para desenhar objetos geométricos simples (ex. pontos, segmentos de linha e polígonos), enquanto outros afetam a renderização destes objetos primitivos que incluem como eles são iluminados ou são coloridos e como eles são mapeados do modelo de espaço bidimensional ou tridimensional do usuário para a tela bidimensional. Há também chamadas que controlam diretamente o framebuffer, como ler e escrever pixels.

1.1 A OpenGL como uma Application Programming Interface (API)

Cientistas e engenheiros foram os primeiros usuários de gráficos 3D. Depois de gastar uma grande quantidade de tempo e esforço, eles descobriram que a linguagem usada para controlar o sistema de apresentação gráfica, freqüentemente, não se integrava com as demais tarefas com as quais eles se deparavam. Isso era mais evidente nos casos da integração de ambientes gráficos de usuário (GUI) e gráficos 3D.

A OpenGL foi projetada para ser neutra ao sistema de janelas e neutra ao fornecedor. Isto significa que outros ambientes Windows, como NT ou Windows 95/98, também suportam a OpenGL. Isso é uma grande vantagem e é possível já que a especificação da OpenGL é, independente de [Shamansky (1995)] :

1. sistema windows
2. sistema operacional
3. rede

A OpenGL é controlada por um consórcio de indústrias, o *OpenGL Architectural Review Board*, um comitê formado pelos maiores fabricantes mundiais de hardware e software, que controla o desenvolvimento da OpenGL, assegurando assim que nenhum fabricante sozinho imponha uma direção à OpenGL.

A OpenGL não é [Shamansky, (1995)]:

- um kit de ferramentas ou API de alto nível;
- um sistema windows;
- um sistema descritivo de gráficos;
- orientada a objetos.

A OpenGL é dependente de um sistema windows para realizar tarefas relacionadas a janela (como criar a tela, lidar com o dados de entrada do usuário, etc.). Ela não é *descritiva*, isto é, o programador não monta um modelo da cena a ser renderizada, e então deixa o sistema gráfico se encarregar da tarefa de fazer o desenho.

A OpenGL é [Shamansky, (1995)]:

- um sistema de modo imediato;
- flexível ao formato da aplicação;
- um sistema procedural;
- funcional a listas de displays.

A OpenGL é um sistema de modo imediato cujos comandos são executados, na maioria das vezes, imediatamente. A OpenGL é flexível porque aceita vários tipos de dados, livrando o programador de conversões desnecessárias. É procedural porque o programador escreve comandos específicos para determinar o que é realmente desenhado. E suporta listas de displays não-editáveis para uma melhor extensibilidade da rede, resultando em performance melhor para aplicações cliente/servidor, como também uma renderização direta melhor.

2. Mapeamento de Texturas

A OpenGL oferece funções de mapeamento de texturas que encaixam imagens dentro de polígonos na cena. É função do programador lidar com a aplicação dessas imagens dentro dos polígonos.

O mapeamento de texturas é usado em jogos, tais como Quake, para imagens realistas de salas e personagens. Diferente da OpenGL, esses jogos usam um método de aplicação de textura chamado *raycasting* para mapear imagens de textura dentro dos polígonos. Apesar da técnica de *raycasting* ser muito mais rápida em placas gráficas comuns do que o mapeamento de texturas oferecido pela OpenGL, é limitada a superfícies planas em um plano 2D. Isto é, não se pode olhar de cima ou de baixo. Mapeamento de texturas em OpenGL não tem essa limitação, mas se pode esperar que funcione mais lentamente em placas gráficas comuns.

Algumas placas aceleradoras de gráficos 3D mais recentes e mais baratas suportam OpenGL e aplicação de texturas via hardware. Quando uma placa suporta mapeamento de texturas via hardware, a CPU não precisa fazer todos os cálculos para o mapeamento e preparação para a aplicação da textura - a placa desempenha esse papel.

2.1 Princípios de Mapeamento de Texturas

O mapeamento de texturas em OpenGL é razoavelmente direto. Para começar, cada textura é uma imagem de algum tipo.

Uma textura 1D é uma imagem com largura mas sem altura, ou vice-versa; texturas 1D possuem largura ou altura de 1 pixel. Pode-se imaginar que texturas 1D não são muito úteis, mas elas realmente podem tomar o lugar de técnicas mais comuns de sombreamento de cores e acelerarem a renderização no processo. A figura 1 mostra uma textura 1D “ROY-G-BIV” (Red, Orange, Yellow - Green - Blue, Indigo, Violet) para exibir um arco-íris. A imagem textura é uma linha de pixels (valores de cor) cobrindo o espectro de cores visto em um arco-íris. A cena equivalente sem textura deveria conter sete vezes mais polígonos da imagem com textura e necessitaria muito mais tempo de renderização.

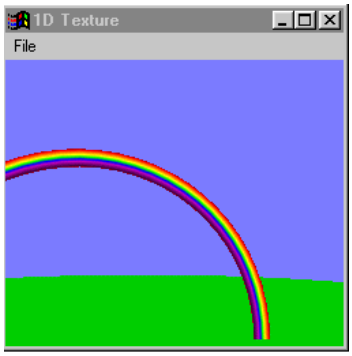

Figura 1 - Um arco-íris de textura 1D

Uma textura 2D é uma imagem que possui mais de um pixel de largura e altura e é geralmente carregada de um mapa de bits (arquivo com extensão .BMP no Windows). Texturas bidimensionais são normalmente usadas para substituir superfícies de geometria complexa (muitos polígonos) em edifícios, árvores, etc. Estas texturas 2D também podem ser usadas para acrescentar detalhes realistas de cenário, como as nuvens no céu na figura 2.

Figura 2 - A textura 2D de um céu e a cena resultante

As texturas 1D e 2D vistas até agora são compostas de valores de cores RGB. As texturas também podem ser compostas de índices de cores ou níveis de luminância (cinza), e podem incluir valores de alpha (transparência). Este último é útil para definir objetos naturais, como árvores, porque o valor alpha pode ser usado para fazer a árvore visível mas deixar o cenário ser mostrado através dela. Os valores de alpha podem variar de 0.0 (transparência completa) até 1.0 (nenhuma transparência, objeto opaco).

Alguns equipamentos também suportam texturas 3D (volume) com OpenGL. Texturas de volumes são usadas para visualização de imagens CAT, MRI e outras 3D. Infelizmente, até mesmo uma imagem de textura pequena de 256 x 256 x 256 em tons de cinza irá necessitar de mais de 16 Mbytes de memória.

2.2 Definindo Imagens de Textura

Imagens de textura seguem os mesmos princípios de armazenamento dos mapas de bits.

O padrão OpenGL exige que as dimensões das imagens de texturas devam ser em potências de 2. As imagens de textura podem também ter 1 ou dois pixels de borda sobre suas extremidades para definir a cor dos polígonos que estão fora da imagem de textura.

2.2.1 Definindo Texturas 1D

A OpenGL possui uma única função para definição de texturas 1D: `glTexImage1D`. A função `glTexImage1D` aceita oito argumentos:

```
void glTexImage1D(GLenum target,
      GLint level,
      GLint components,
      GLsizei width,
      GLint border,
      GLenum format,
      GLenum type,
      const GLvoid *pixels)
```

O argumento `target` especifica qual textura deve ser definida; este argumento deve ser `GL_TEXTURE_1D`. O argumento `level` indica o nível de detalhe da imagem e normalmente é zero. Outros valores são usados para texturas *mipmapped*. O argumento `components` especifica o número de valores de cor usados para cada pixel. Para texturas de índice de cores, `components` deve ser 1. Valores 3 e 4 são usados, respectivamente, para texturas de imagens RGB e RGBA.

`Width` e `border` especificam o tamanho da imagem de textura. O valor `border` controla o número de pixels de borda que a OpenGL deveria respeitar (e usar) e pode ser um valor entre 0, 1 ou 2. O parâmetro `width` especifica a lagura da imagem de textura principal (sem os pixels de borda) e deve ser uma potência de 2.

O argumento `format` indica o tipo de valores de cor a respeitar - `GL_COLOR_INDEX`, `GL_LUMINANCE`, `GL_RGB` ou `GL_RGBA`.

Listagem 1 - Trecho de código para definição de imagem de textura 1D

```
/*
 *  'LoadAllTextures()' - Carrega as
imagens de textura para a cena.
 */

void
LoadAllTextures(void)
{
  static        unsigned        char
roygbiv_image[8][3] =
  {
    { 0x3f, 0x00, 0x3f },     /*  Dark
Violet (para 8 cores...) */
    { 0x7f, 0x00, 0x7f },     /*
Violet */
    { 0xbf, 0x00, 0xbf },     /*
Indigo */
    { 0x00, 0x00, 0xff },     /*  Blue
*/
    { 0x00, 0xff, 0x00 },     /* Green
*/
    { 0xff, 0xff, 0x00 },     /*
Yellow */
    { 0xff, 0x7f, 0x00 },     /*
Orange */
    { 0xff, 0x00, 0x00 }      /*   Red
*/
  };

  glNewList(RainbowTexture          =
glGenLists(1), GL_COMPILE);
    glTexParameteri(GL_TEXTURE_1D,
GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_1D,
GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage1D(GL_TEXTURE_1D, 0, 3,
8,   0,   GL_RGB,   GL_UNSIGNED_BYTE,
roygbiv_image);
  glEndList();
}
```

O código da listagem 1 cria uma lista de exibição contendo a imagem de textura e o filtro de aumento e redução, `GL_LINEAR`. O filtro de redução é usado quando o polígono a ser desenhado é menor que a imagem de textura, neste caso 8 pixels. O filtro de aumento é usado quando o polígono é maior que a imagem de textura. Designando o filtro `GL_LINEAR`, é dito à OpenGL para interpolar linearmente valores de cor na imagem de textura antes de desenhar qualquer coisa na tela. Os outros filtros que podem ser usados para `GL_TEXTURE_MIN_FILTER` são apresentados na tabela 1.

Tabela 1 - Filtros de Imagens de Textura

Filtro	Descrição
`GL_NEAREST`	Filtro do vizinho mais próximo.
`GL_LINEAR`	Interpolação linear.
`GL_NEAREST_MIPMAP_NEAREST`	Filtragem de vizinho próximo *mipmapped.*
`GL_NEAREST_MIPMAP_LINEAR`	*Mipmaps* interpolados linearmente.
`GL_LINEAR_MIPMAP_NEAREST`	Interpolação linear de *mipmaps.*
`GL_LINEAR_MIPMAP_LINEAR`	*Mipmaps* interpolados de interpolação linear

A filtragem `GL_NEAREST` pega o pixel mais próximo na imagem de textura em vez da interpolação entre pixels.

2.2.2 Definindo Texturas 2D

Para se definir uma imagem de textura 2D em OpenGL, chama-se a função `glTexImage2D`. A função

`glTexImage2D` recebe um argumento de altura, além dos argumentos usados na função `glTexImage1D`:

```
void glTexImage2D(GLenum target,
      GLint level,
      GLint components,
      GLsizei width,
      GLsizei height,
      GLint border,
      GLenum format,
      GLenum type,
      const GLvoid *pixels)
```

Como a função `glTexImage1D`, os argumentos `width` e `height` devem ser potências de 2. A listagem 2 mostra como carregar uma imagem de textura completa do céu com nuvens:

Listagem 2 - Trecho de código para definição de imagem de textura 2D

```
/*
 *  'LoadAllTextures()' - Carrega as
imagens de textura para a cena.
 */

void
LoadAllTextures(void)
{
  BITMAPINFO     *info;
            /* Bitmap information */
  void           *bits;
            /* Bitmap pixel bits */
  GLubyte   *rgb;
            /* Bitmap RGB pixels */

 /*
  * Tenta carregar o bitmap e
converte-lo para RGB...
  */

  bits = LoadDIBitmap("sky.bmp",
&info);
  if (bits == NULL)
    return;

  rgb = ConvertRGB(info, bits);
  if (rgb == NULL)
  {
    free(info);
    free(bits);

    return;
  };

  glNewList(SkyTexture =
glGenLists(1), GL_COMPILE);
    glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_NEAREST);

   /*
    * Define a imagem textura 2D.
    */

   glPixelStorei(GL_UNPACK_ALIGNMENT,
4);/* Forca alinhamento de 4 bytes */

glPixelStorei(GL_UNPACK_ROW_LENGTH,
0);
glPixelStorei(GL_UNPACK_SKIP_ROWS,
0);
glPixelStorei(GL_UNPACK_SKIP_PIXELS,
0);

    glTexImage2D(GL_TEXTURE_2D, 0, 3,
info->bmiHeader.biWidth, info-
>bmiHeader.biHeight, 0, GL_RGB,
GL_UNSIGNED_BYTE, rgb);
  glEndList();

 /*
  * Libera o bitmap e as imagens RGB,
entao retorna 0 (sem erros).
  */

  free(rgb);
  free(info);
  free(bits);
}
```

2.3 Desenhando Polígonos com Textura

Uma vez definida uma textura, ainda se deve habilitar a aplicação da textura. Para se habilitar a aplicação da textura, deve-se usar as seguintes chamadas:

```
glDisable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_1D);
glTexEnvi(GL_TEXTURE_ENV,
GL_TEXTURE_ENV_MODE, GL_DECAL);
```

A chamada `glEnable` habilita a aplicação de textura 1D. Se a chamada à função que habilita não for feita, nenhum dos polígonos será texturizado. A função `glTexEnvi` ajusta a aplicação de textura para o modo "decalco", significando que as imagens serão aplicadas diretamente sobre os polígonos.

Outros modos de aplicação de texturas são listados na Tabela 2.

Tabela 2 - Modos de textura para `GL_TEXTURE_ENV_MODE`

Modo	Descrição
`GL_MODULATE`	Os pixels da textura "filtram" os pixels existentes na tela.
`GL_DECAL`	Os pixels da textura substituem os

	pixels existentes na tela.
GL_BLEND	Os pixels da textura "filtram" as cores existentes dos pixels na tela e são combinados com uma cor constante.

O modo de textura GL_MODULATE multiplica a cor atual da textura (ou luminância) pela cor na tela. Para texturas de somente um componente (luminância), isto se torna um filtro de brilho que irá variar o brilho da imagem da tela baseado na imagem de textura. Para texturas de três componentes (RGB), pode-se gerar efeitos de filtro de lentes coloridas.

Diferente do modo GL_MODULATE, a texturização GL_BLEND permite que se misture uma cor constante na cena baseado na imagem de textura. O modo GL_BLEND é usado para texturizar objetos como nuvens; a cor constante deveria ser o branco e a imagem de textura seria uma nuvem.

Uma vez definido o modo de texturização a ser usado, pode-se fazer o desenho dos polígonos. A listagem 3 mostra como desenhar o arco-íris da Figura 1.

Listagem 3 - Trecho de código para desenho de um arco-íris texturizado

```
    glEnable(GL_TEXTURE_1D);
    glCallList(RainbowTexture);
    glBegin(GL_QUAD_STRIP);
      for (th = 0.0; th <= M_PI; th
+= (0.03125 * M_PI))
      {
      x = cos(th) * 50.0;
      y = sin(th) * 50.0;
      z = -50.0;
      glTexCoord1f(0.0);
      glVertex3f(x, y, z);

      x = cos(th) * 55.0;
      y = sin(th) * 55.0;
      z = -50.0;
      glTexCoord1f(1.0);
      glVertex3f(x, y, z);
      };
    glEnd();
```

Para posicionar a textura ROY-G-BIV no arco-íris, chama-se glTexCoord. Para texturas 1D, chama-se uma das funções glTexCoord1f, glTexCoord1d, glTexCoord1s ou glTexCoordli. Um valor de 0.0 representa o pixel mais à esquerda na imagem, e 1.0 representa o pixel mais à direita. Valores fora dessa faixa são manipulados diferentemente dependendo do valor do parâmetro GL_TEXTURE_WRAP_S. Se GL_TEXTURE_WRAP_S está setado para GL_CLAMP (o default), então as coordenadas da textura ficam restritas a um intervalo de 0.0 a 1.0, inclusive. Quando um polígono se perde da imagem de textura, ele é desenhado usando-se a(s) cor(es) sobre as bordas da imagem de textura, ou as cores da borda da imagem da textura, se definidas. As coordenadas de textura são tradicionalmente chamadas de S e T (ou s,t), em vez de X e Y.

Se for usada GL_REPEAT, a imagem de textura é ladrilhada sobre o polígono. As coordenadas da textura são módulo 1.0, isto é, a imagem de textura se repete em intervalos regulares. O modo de aplicação de textura GL_REPEAT pode ser usado para reduzir o tamanho das imagens de textura em superfícies repetitivas. O desafio com estes tipos de textura é fazer com que os vértices de cada ladrilho combinem com os próximos.

2.4 Texturas com muitas imagens (Mipmapped Textures)

Até agora, trabalhou-se exclusivamente com imagens de textura simples. Isto é, quando um polígono texturizado é desenhado, o polígono é pintado como uma imagem 1D ou 2D. Isto é suficiente para algumas cenas, mas exibições animadas normalmente precisam de vários níveis de detalhe dependendo da distância do observador. Por exemplo, quando caminhamos através de uma sala virtual, você poderia querer uma imagem com uma alta resolução de perto mas somente um rascunho a uma distância maior.

A OpenGL suporta texturas com múltiplas imagens, chamadas texturas *mipmapped*. A técnica *mipmapping* seleciona a imagem de textura para um polígono mais próximo da resolução da tela. A carga de texturas *mipmapped* leva um pouco mais de tempo que texturas padrão, mas os resultados visuais são impressionantes. Além disso, as texturas *mipmapped* podem melhorar a performance de exibição reduzindo a necessidade de filtros de imagem GL_LINEAR.

As texturas *mipmapped* são definidas fornecendo-se um parâmetro específico de nível para cada imagem.

Os níveis de imagem são especificados no primeiro parâmetro da chamada glTexImage1D(). A imagem nível 0 é a primária, com a maior resolução para a textura. A imagem nível 1 é metade do tamanho da imagem primária e assim por diante. Quando se desenha polígonos com uma textura *mipmapped*, é necessário o uso de filtros redutores (GL_TEXTURE_MIN_FILTER), na tabela 3:

Tabela 3 - Filtros redutores

Filtro	Descrição
GL_NEAREST_MIPMAP_NEAREST	Usa a imagem mais próxima da resolução da tela (polígono). Usa o filtro GL_NEAREST quando faz a texturização com esta imagem.
GL_NEAREST_MIPMAP_LINEAR	Usa a imagem mais próxima da resolução da tela (polígono). Usa o filtro GL_LINEAR quando faz a texturização com esta imagem.
GL_LINEAR_MIPMAP_NEAREST	Interpola linearmente entre duas imagens mais próximas da resolução tela (polígono). Usa o filtro GL_NEAREST quando faz

	a texturização com esta imagem.
GL_LINEAR_MIP MAP_LINEAR	Interpola linearmente entre duas imagens mais próximas da resolução tela (polígono). Usa o filtro GL_LINEAR quando faz a texturização com esta imagem.

2.5 Definindo Texturas 3D

O comando

```
void TexImage3D( enum target,
   int level, int internalformat,
   sizei width, sizei height,
   sizei depth, int border,
   enum format, enum type,
   void *data );
```

é usado para especificar uma imagem de textura tridimensional. *target* deve ser **TEXTURE_3D** ou **PROXY_TEXTURE_3D**. *format*, *type*, e *data* são os mesmos argumentos correspondentes para **DrawPixels**; eles especificam o formato dos dados da imagem, o tipo desses dados, e um ponteiro para os dados de imagem em memória.

Os grupos em memória são tratados como sendo organizados em uma sucessão de retângulos adjacentes. Cada retângulo é uma imagem bidimensional cujo tamanho e organização é especificado pelos parâmetros *width* e *height* para **TexImage3D**. Os valores de UNPACK_ROW_LENGTH e UNPACK_ALIGNMENT controlam o espaçamento linha a linha nestas imagens da mesma maneira que a função **DrawPixels**. Se o valor do parâmetro inteiro UNPACK_IMAGE_HEIGHT não é positivo, então, o número de linhas em cada imagem bidimensional é igual a *height*; caso contrário o número de linhas é UNPACK_IMAGE_HEIGHT. Cada imagem bidimensional inclui um número inteiro de linhas, e é exatamente adjacente às suas imagens vizinhas.

2.6 Comandos Alternativos de Especificação de Imagens Textura

Imagens textura 1D e 2D também podem ser especificadas usando-se dados de imagens retirados diretamente do framebuffer, e subregiões retangulares de imagens textura podem ser especificadas.

O comando

```
void CopyTexImage2D( enum target, int
      level, enum internalformat, int
      x, int y, sizei width, sizei
      height, int border );
```

define um vetor bidimensional exatamente da mesma maneira que **TexImage2D**, exceto que os dados da imagem são retirados do framebuffer em vez da memória cliente.

Seis comandos adicionais,

```
void TexSubImage3D( enum target, int
      level, int xoffset, int
      yoffset, int zoffset, sizei
      width, sizei height,
      sizei depth, enum format, enum
      type, void *data );

void TexSubImage2D( enum target, int
      level, int xoffset, int
      yoffset, sizei width, sizei
      height, enum format, enum type,
      void *data );

void TexSubImage1D( enum target, int
      level, int xoffset, sizei
      width, enum format, enum type,
      void *data );

void CopyTexSubImage3D( enum target,
      int level, int xoffset, int
      yoffset, int zoffset, int x,
      int y, sizei width, sizei
      height );

void CopyTexSubImage2D( enum target,
      int level, int xoffset, int
      yoffset, int x, int y, sizei
      width, sizei height );

void CopyTexSubImage1D( enum target,
      int level, int xoffset, int
      x, int y, sizei width );
```

reespecificam somente uma subregião retangular de um vetor de textura existente.

Cada um dos comandos **TexSubImage** interpreta e processa grupos de pixels exatamente da mesma maneira que o seu oposto **TexImage**, exceto que a atribuição dos valores R, G, B e a de grupos de pixels para os componentes da textura é controlado pelo parâmetro *internalformat* do vetor de textura, não por um argumento do comando.

2.7 Parâmetros de Textura

Vários parâmetros controlam como o vetor de textura é tratado quando aplicado em um fragmento. Cada parâmetro é ajustado chamando-se:

```
void TexParameter{if}( enum target,
      enum pname, T param );
void TexParameter{if}v( enum target,
      enum pname, T params );
```

target é o alvo, tanto **TEXTURE_1D**, **TEXTURE_2D** ou **TEXTURE_3D**. *pname* é uma constante simbólica indicando o parâmetro a ser setado. Na primeira forma do comando, *param* é um valor que ajusta apenas um parâmetro; na segunda forma do comando, *param* é um vetor de parâmetros cujos tipos dependem do parâmetro sendo ajustado.

2.8 Objetos Textura

Além das texturas default **TEXTURE_1D**, **TEXTURE_2D** e **TEXTURE_3D**, objetos textura denominados 1D, 2D e 3D podem ser criados e manipulados. O espaço de nomes para objetos textura

são os inteiros sem sinal, com o zero reservado pela OpenGL.

Um objeto textura é criado atribuindo-se um nome não utilizado a **TEXTURE_1D**, **TEXTURE_2D** ou **TEXTURE_3D.** A atribuição é efetuada chamando-se o comando

```
void BindTexture (enum target, uint texture );
```

com *target* setado para o alvo textura desejado e *texture* ajustado para o nome não usado. O objeto textura resultante é um novo vetor de estado, compreendendo todos os valores de estado setados para seus valores iniciais. Se o novo objeto textura é colado a **TEXTURE_1D**, **TEXTURE_2D** ou **TEXTURE_3D**, ele é e permanece sendo uma textura 1D, 2D ou 3D, respectivamente, até ser removido.

Objetos textura são removidos chamando-se

```
void DeleteTexture (sizei n, uint *textures );
```

onde *n* contém "n" nomes de objetos textura a serem removidos.

O comando

```
void GenTexture (sizei n, uint *textures );
```

retorna *n* nome de objetos textura previamente não usados em *textures*.

2.9 Aplicação de Textura

A aplicação de textura é habilitada ou desabilitada usando-se os comandos genéricos **Enable** e **Disable**, respectivamente, com as constantes simbólicas **TEXTURE_1D**, **TEXTURE_2D** ou **TEXTURE_3D** para habilitar a textura 1D, 2D ou 3D, respectivamente. Se ambas as aplicações de textura 2D e 1D estão habilitadas, a aplicação de textura bidimensional é usada. Se a aplicação de textura 3D for habilitada, esta tem prioridade sobre os demais modos. Se toda a texturização for desabilitada, um fragmento rasterizado é passado sem alteração para o próximo estágio da OpenGL (mesmo que suas coordenadas de textura possam ser descartadas). Por outro lado, um valor de textura é encontrado de acordo com os valores de parâmetros da atual imagem textura colada das mesmas dimensões. Este valor de textura é usado juntamente com o fragmento que vem da computação da função de textura indicado pelo atual ambiente de textura. O resultado desta função troca os valores R, G, B e A primários do fragmento que vem. Esses são os valores de cor passados para as operações subseqüentes. Outros dados associados com o fragmento que vem permanecem inalterados, exceto que as coordenadas de textura podem ser descartadas.

3. Exemplos de Imagens com aplicação de textura através do uso da OpenGL

Figura 3 - Visualização de terrenos em 3D [Wright Jr--Sweet (1996)]

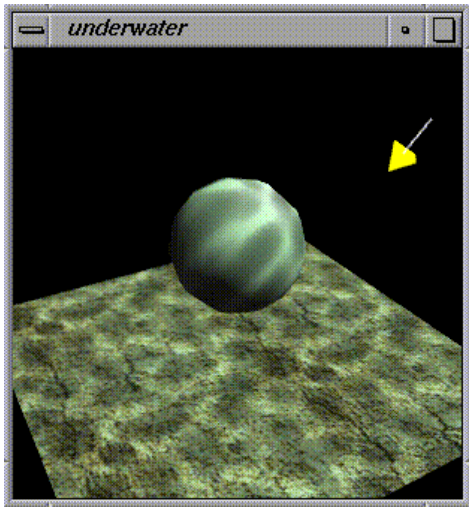

Figura 4 - Cena de uma esfera na água [Kilgard (1997b)]

Conclusão

Neste artigo buscou-se apresentar alguns tópicos relevantes na utilização da API OpenGL na geração de imagens por computador. Nota-se que OpenGL é um padrão em franca expansão, a qual está sendo adotada como padrão de programação de jogos 3D, principalmente. A aplicação de texturas a fim de se obter realismo é uma técnica presente na maioria dos softwares que lidam com apresentação de objetos 3D, com a intenção de proporcionar ao usuário uma sensação de estar presente na realidade que o jogo busca alcançar.

Nota-se que, a medida em que são desenvolvidas novas placas aceleradoras de gráficos para utilização em PC's domésticos, também se aceleram as pesquisas rumo a obtenção de situações de realidade virtual em que o usuário pensa estar imerso no ambiente.

Uma outra API para geração de imagens 3D busca se tornar padrão, a Direct3D. Porém a OpenGL se mostra mais adequada para suporte à renderização rápida de reflexos e sombras. Além de possuir uma boa documentação, a OpenGL se mostra uma API mais "limpa" e mais fácil de ser aprendida.

Há uma tendência de muitos fabricantes de hardware gráfico em oferecer suporte à OpenGL em seus produtos.

Referências:

J. de M. Gomes, S. Sandri, L. Velho, *Instructions for Authors - SIBGRAPI.* 1998.

M. Kilgard. *Shadows, Reflections, Lighting, Textures. Easy with OpenGL!* 1997a. Disponível por WWW em http://reality.sgi.com/opengl/tips/TexShadowReflectLight.html (13 Nov. 1998)

M. Kilgard. *OpenGL-rendering of Underwater Caustics.* 1997b. Disponível por WWW em http://reality.sgi.com/opengl/tips/caustics/ (13 Nov. 1998)

M. Segal, K. Akeley, J. Leech. *The OpenGL Graphics System: A Specication (Version 1.2.1).* 1998. Disponível por WWW em ftp://sgigate.sgi.com/pub/opengl/doc/opengl1.2/opengl1.2.1.pdf (27 Dez. 1998)

H. Shamansky. *OpenGL - The Integration of Windowing and 3D Graphics***.** 1995. Disponível por WWW em http://hertz.eng.ohio-state.edu/~hts/opengl/article.html (23 Nov. 1998)

R. Wright Jr., M. Sweet. *OpenGL Superbible***,** Corte Madera: Waite Group Press, 1996.