Fontes baseadas em textura para OpenGL ES

Aquivado: Este artigo foi arquivado, pois o conteúdo não é mais considerado relevante para se criar soluções comerciais atuais. Se você achar que este artigo ainda é importante, inclua o template {{ForArchiveReview|*escreva a sua justificativa*}}.

Acredita-se que este artigo ainda seja válido no contexto original (quando ele foi escrito)

Uma abordagem eficiente para se implementar fontes em OpenGL [1] é utilizar um quadrilátero com textura para cada caracter. Essa técnica possui desempenho muito bom e apresenta bons resultados visuais. Para maiores detalhes sobre esse assunto, por favor consulte este endereço.

glFont

A ferramenta glFont [2] é uma conhecida aplicação para gerar texturas contendo um conjunto de caracteres, e uma API para desenhá-los. De acordo com a sua licença, essa ferramenta é gratuita para usos comerciais ou não-comerciais. Mais detalhes sobre a licença (Inglês).

O autor original é Brad Fish (brad.fish@gmail.com).

Aqui está uma conversão da classe original, para ser usada no Symbian OS (o autor gentilmente permitiu que a conversão fosse publicada neste site).

```
//*****************************************************************
// glfont2.h -- Header for glfont2.cpp
// Copyright (c) 1998-2002 Brad Fish
// See glfont.html for terms of use
// May 14, 2002
//
// Symbian OS port - June 2007
// Luis Valente - lpvalente no gmail.com
//
//*****************************************************************

#ifndef GLFONT2_H
#define GLFONT2_H

#include <e32base.h>
#include <GLES/gl.h>

//_____________________________________________________________________________
//
// Classe simples para desenhar texto na forma de quadriláteros
// com mapeamento de textura. Não é possível usar texto Unicode
// no momento. O ponto (0,0) do caracter fica no topo-esquerdo
// do quadrilátero.

class GLFont
{
  public:

   /**
    * Criar uma instância da classe, a partir de um arquivo gerado pela
    * ferramenta.
    */
   static GLFont* NewL (const TDesC & aFilename);

  public:

   /**
    * Destrutor.
    */
   ~GLFont ();

  public:


   /**
    * Recuperar a largura e altura da textura.
    */
   void GetTexSize (TInt & aWidth, TInt & aHeight);

   /**
    * Recuperar o intervalo de caracteres presente.
    */
   void GetCharInterval (TInt & aStart, TInt & aEnd);

   /**
    * Recuperar as dimensões do caracter.
    */
   void GetCharSize (TText8 c, TInt & aWidth, TInt aHeight);


   /**
    * Calcular o tamanho de uma string.
    */
   void GetStringSize (const TDesC8 & aText, TInt & aWidth, TInt & aHeight);

   /**
    * Desenahr o texto.
    */
   void DrawString (const TDesC8 & aText, GLfixed aX, GLfixed aY);

   /**
    * Ligar os estados necessários para o desenho.
    */
   void BeginDraw ()
   {
    glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable (GL_TEXTURE_2D);
```


(C) Copyright Nokia 2013. All rights reserved.

```
    glEnableClientState (GL_TEXTURE_COORD_ARRAY);
   }

   /**
    * Desligar os estados necessários.
    */
   void EndDraw ()
   {
    glDisable (GL_BLEND);
    glDisable (GL_TEXTURE_2D);
    glDisableClientState (GL_TEXTURE_COORD_ARRAY);
   }

 private:

    /**
     * Construtor padrão.
     */
   GLFont ();

   /**
    * Segunda parte do construtor de duas fases.
    */
   void ConstructL (const TDesC & aFilename);

   /**
    * Carregar o arquivo de fonte.
    */
   void LoadFileL (RFs & aFs, const TDesC & aFilename);

   /**
    * Destruir a fonte.
    */
   void Destroy ();


  private:

   // caracter simples
   struct GLFontChar
   {
      GLfixed dx, dy;
      GLfixed tx1, ty1;
      GLfixed tx2, ty2;
   };

   // cabeçalho do arquivo de fonte.
   struct GLFontHeader
   {
      GLuint tex;
      TInt   texWidth, texHeight;
      TInt   startChar, endChar;
      GLFontChar *chars;
   };

 private:

   GLFontHeader iHeader;
};

//*********************************************************************
#endif
```

Aqui está a implementação da classe.

```
//*********************************************************************
// glfont2.cpp -- glFont Version 2.0 implementation
// Copyright (c) 1998-2002 Brad Fish
// See glfont.html for terms of use
// May 14, 2002
//
// Symbian OS port - June 2007
// Luis Valente - lpvalente@gmail.com
//
//*********************************************************************

// Symbian OS headers
#include <s32file.h>
#include <eikenv.h>
#include <eikappui.h>
#include <eikapp.h>
#include "glfont2.h"
#include "FixedMath.h"


// estrutura GLFontChar que está no arquivo
struct GLFontCharFile
{
   TReal32 dx, dy;
   TReal32 tx1, ty1;
   TReal32 tx2, ty2;
};

// estrutura GLFontHeaderFile como está no arquivo
struct GLFontHeaderFile
{
   TInt32  tex;
   TInt32  texWidth, texHeight;
   TInt32  startChar, endChar;
   TUint32 chars;
};



//_____________________________________________________________________________
//
// Construtor padrão.
//

GLFont::GLFont ()
{
   // Inicializar iHeader para um estado seguro
   iHeader.tex = 0;
   iHeader.texWidth = 0;
   iHeader.texHeight = 0;
   iHeader.startChar = 0;
   iHeader.endChar = 0;
```


(C) Copyright Nokia 2013. All rights reserved.

```
   iHeader.chars = NULL;

   // textura da OpenGL
   glGenTextures (1, &iHeader.tex);
}

//_____________________________________________________________________________
//
// Destrutor.
//

GLFont::~GLFont ()
{
   // Destruir a fonte
   Destroy();

   // apagar texture
   glDeleteTextures (1, &iHeader.tex);
}

//_____________________________________________________________________________
//
// Criar uma instância da classe, lendo um arquivo de fonte.
//

GLFont * GLFont::NewL (const TDesC & aFilename)
{
   GLFont* f = new (ELeave) GLFont();
   CleanupStack::PushL (f);

   f->ConstructL (aFilename);

   CleanupStack::Pop ();
   return f;
}

//_____________________________________________________________________________
//
// Segunda parte do construtor de duas fases.
//

void GLFont::ConstructL (const TDesC & aFilename)
{
   // Destruir a fonte anterior, se existir
   Destroy();

   // Abrir sessão com o servidor de arquivos
   RFs session;
   User::LeaveIfError (session.Connect());
   CleanupClosePushL (session);

   // Recuperar o diretório privado da aplicação
   TFileName path;
   session.PrivatePath (path);

   // Recuperar o caminho completo da aplicação no dispositivo
   #ifndef __WINS__
      TFileName appFullName =
      CEikonEnv::Static()->EikAppUi()->Application()->AppFullName();

      TParse parse;
      parse.Set (appFullName, NULL, NULL);
      path.Insert (0, parse.Drive());
   #endif

   // Atualizar o nome de arquivo com o caminho completo
   TFileName fullFilename (path);
   fullFilename.Append (aFilename);

   // Carregar arquivo
   LoadFileL (session, fullFilename);

   // Fechar a sessão com o servidor.
   CleanupStack::PopAndDestroy();

}

//_____________________________________________________________________________
//
// Carregar o arquivo de fonte.
//

void GLFont::LoadFileL (RFs & aFs, const TDesC & aFilename)
{
   // Abrir o arquivo de entrada
   RFileReadStream readStream;

   User::LeaveIfError (readStream.Open (aFs, aFilename, EFileRead));
   readStream.PushL();

   // Ler o cabeçalho
   GLFontHeaderFile headerFile;

   headerFile.tex       = readStream.ReadInt32L ();
   headerFile.texWidth  = readStream.ReadInt32L();
   headerFile.texHeight = readStream.ReadInt32L();
   headerFile.startChar = readStream.ReadInt32L();
   headerFile.endChar   = readStream.ReadInt32L();
   headerFile.chars     = readStream.ReadUint32L();

   // Copiar iHeader para a versão em memória
   iHeader.texWidth  = headerFile.texWidth;
   iHeader.texHeight = headerFile.texHeight;
   iHeader.startChar = headerFile.startChar;
   iHeader.endChar   = headerFile.endChar;

   // Alocar espaço para o array de caracteres
   TInt numChars = iHeader.endChar - iHeader.startChar + 1;
   iHeader.chars = new (ELeave) GLFontChar [numChars];

   // Ler dados desse array
   for (TInt i = 0; i < numChars; ++i)
   {
      iHeader.chars [i].dx  =  FloatToFixed (readStream.ReadReal32L () );
      iHeader.chars [i].dy  =  FloatToFixed (readStream.ReadReal32L () );
      iHeader.chars [i].tx1 =  FloatToFixed (readStream.ReadReal32L () );
      iHeader.chars [i].ty1 =  FloatToFixed (readStream.ReadReal32L () );
      iHeader.chars [i].tx2 =  FloatToFixed (readStream.ReadReal32L () );
      iHeader.chars [i].ty2 =  FloatToFixed (readStream.ReadReal32L () );
   }
```


(C) Copyright Nokia 2013. All rights reserved.

```
   }

   // Ler dados da textura com os caracteres
   TInt numTexBytes = iHeader.texWidth * iHeader.texHeight * 2;
   TUint8 * texBytes = new (ELeave) TUint8 [numTexBytes];
   CleanupStack::PushL (texBytes);

   readStream.ReadL (texBytes, numTexBytes);

   // Criar a textura da OpenGL
   glBindTexture   (GL_TEXTURE_2D, iHeader.tex);
   glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
   glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
   glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

   glTexImage2D (GL_TEXTURE_2D,
                 0,
                 GL_LUMINANCE_ALPHA,
                 iHeader.texWidth,
                 iHeader.texHeight,
                 0,
                 GL_LUMINANCE_ALPHA,
                 GL_UNSIGNED_BYTE,
                 (GLvoid *)texBytes);

   // Liberar memória alocada para os dados da textura
   CleanupStack::Pop ();
   delete [] texBytes;

   // Fechar arquivo
   readStream.Close();
   readStream.Pop();
}

//____________________________________________________________________________
//
// Destruir a fonte.
//

void GLFont::Destroy ()
{
   // Apagar o array de caracteres caso seja necessário
   if (iHeader.chars)
   {
      delete [] iHeader.chars;
      iHeader.chars = 0;
   }
}

//____________________________________________________________________________
//
// Recuperar dimensões da textura.
//

void GLFont::GetTexSize (TInt & aWidth, TInt & aHeight)
{
   aWidth  = iHeader.texWidth;
   aHeight = iHeader.texHeight;
}

//____________________________________________________________________________
//
// Recuperar o intervalo de caracteres disponível.
//

void GLFont::GetCharInterval (TInt & aStart, TInt & aEnd)
{
   aStart = iHeader.startChar;
   aEnd   = iHeader.endChar;
}

//____________________________________________________________________________
//
// Recuperar as dimensões de um determinado caracter.
//

void GLFont::GetCharSize (TText8 aChar, TInt & aWidth, TInt aHeight)
{
   // Certificar-se de que o caracter está na faixa válida.
   if (aChar < iHeader.startChar || aChar > iHeader.endChar)
   {
      // Não é valido, então não tem tamanho
      aWidth  = 0;
      aHeight = 0;
   }
   else
   {
      GLFontChar* fontChar;

      // Recuperar o tamanho do caracter
      fontChar = & iHeader.chars [aChar - iHeader.startChar];
      aWidth  = FixedToInt (MultiplyFixed (fontChar->dx, IntToFixed (iHeader.texWidth) ) );
      aHeight = FixedToInt (MultiplyFixed (fontChar->dy, IntToFixed  iHeader.texHeight) ) );
   }
}

//____________________________________________________________________________
//
// Recuperar dimensões de uma string.
//

void GLFont::GetStringSize (const TDesC8 & aText, TInt & aWidth, TInt & aHeight)
{
   // Altura é igual para os caracteres, nessa versão
   aHeight = FixedToInt (MultiplyFixed (iHeader.chars [iHeader.startChar].dy,
IntToFixed (iHeader.texHeight) ) );

   // texWidth como ponto-fixo
   const GLfixed texWidthx = IntToFixed (iHeader.texWidth);

   // Calcular a largura da string
   GLfixed widthx = 0;
   for (TInt i = 0; i < aText.Length(); i++)
   {
      //  Certificar-se de que o caracter está na faixa válida
      const TText8 c = aText [i];
      if (c < iHeader.startChar || c > iHeader.endChar)
         continue;
```


(C) Copyright Nokia 2013. All rights reserved.

```
      // Recuperar um ponteiro para o caracter
      const GLFontChar* fontChar = & iHeader.chars [c - iHeader.startChar];

      // Recuperar largura e altura
      widthx += MultiplyFixed (fontChar->dx, texWidthx);
   }

   // Guardar largura
   aWidth = FixedToInt (widthx);
}

//____________________________________________________________________________
//
// Desenhar uma string. O ponto de referência para os caracteres é o canto
// superior esquerdo.
//

void GLFont::DrawString (const TDesC8 & aText, GLfixed aX, GLfixed aY)
{
   // OpenGL ES utiliza vertex arrays
   GLfixed  vertices [4*2];
   GLfixed texCoords [4*2];
   const GLubyte indices [] = {1, 2, 0, 3};

   glVertexPointer  (2, GL_FIXED, 0, vertices);
   glTexCoordPointer (2, GL_FIXED, 0, texCoords);

   // Selecionar texture
   glBindTexture (GL_TEXTURE_2D, iHeader.tex);

   // Percorrer todos os caracteres
   for (TInt i = 0; i < aText.Length(); i++)
   {
      // Certificar-se de que o caracter está na faixa válida
      TText8 c = aText [i];
      if (c < iHeader.startChar || c > iHeader.endChar)
         continue;

      // Recuperar um ponteiro para o caracter
      GLFontChar* fontChar = &iHeader.chars [c - iHeader.startChar];

      // Recuperar largura e altura
      GLfixed width =  MultiplyFixed (fontChar->dx,
                                      IntToFixed (iHeader.texWidth) );
      GLfixed height = MultiplyFixed (fontChar->dy,
                                      IntToFixed (iHeader.texHeight) );

      // Especificar coordenadas de textura
      texCoords [0] = fontChar->tx1; texCoords [1] = fontChar->ty1;
      texCoords [2] = fontChar->tx1; texCoords [3] = fontChar->ty2;

      texCoords [4] = fontChar->tx2; texCoords [5] = fontChar->ty2;
      texCoords [6] = fontChar->tx2; texCoords [7] = fontChar->ty1;

      // e vértices
      vertices [0] = aX;         vertices [1] = aY;
      vertices [2] = aX;         vertices [3] = aY - height;

      vertices [4] = aX + width; vertices [5] = aY - height;
      vertices [6] = aX + width; vertices [7] = aY;

      // Desenhar
      glDrawElements (GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);

      // Próximo caracter
      aX += width;
   }

}
```

E aqui está o código necessário para as operações em ponto-fixo.

```
//-----------------------------
#ifndef FIXED_MATH_H_
#define FIXED_MATH_H_
//-----------------------------

// INCLUDES
#include <e32base.h>
#include <e32std.h>
#include <e32math.h>
#include <GLES/gl.h>

// FUNCTIONS

inline GLfixed IntToFixed (GLint aValue)
{ return aValue << 16; }


inline GLfixed FloatToFixed (GLfloat aValue)
{ return (GLfixed) (aValue * 65536.0f); }


inline GLint FixedToInt (GLfixed aValue)
{ return aValue >> 16; }


inline GLfloat FixedToFloat (GLfixed aValue)
{ return (GLfloat) (aValue * (1 / 65536.0f)); }


inline GLfixed MultiplyFixed (GLfixed op1, GLfixed op2)
{
  TInt64 r = (TInt64)op1 * (TInt64)op2;
  return (GLfixed) (r >> 16);
}


//-----------------------------
#endif
```

Observações e limitações

(C) Copyright Nokia 2013. All rights reserved.

Essa classe de fonte é destinada para se desenhar texto 2D. Por isso, é importante que use uma projeção ortográfica antes de se desenhar. O código a seguir serve como exemplo:

```
// Rect() é um método que retorna o retângulo de desenho atual, por exemplo, o
// método da classe de View. O ponto (0,0) se localiza no canto inferior
// esquerdo.

glViewport (0, 0, Rect().Width(), Rect().Height());

glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  glOrthox ( 0, IntToFixed (Rect().Width()), 0, IntToFixed (Rect().Height()),
            -1, 1);

 glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
```

Os métodos `BeginDraw()` e `EndDraw()` devem ser chamados (ou código equivalente) para se ligar os estados necessários para o desenho. Por exemplo, se a transparência (alpha-blending) não for habilitada, será possível ver o retângulo ocupado pelo caracter.

Atualmente, a classe utiliza somente descritores de 8 bits. Futuramente, deve-se extender essa classe para usar descritores de 16 bits e tambem strings provenientes de arquivos de recursos.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.