

ID	...	Creation date	10 de Março, 2009
Platform	S60 3rd Edition	Tested on devices	Nokia N95 8GB
Category	Python	Subcategory	OpenGL ES

Keywords (APIs, classes, methods, functions): opengl es

Introdução

Usar texturas para representar texto em aplicações gráficas 2D e 3D é uma técnica antiga, mas é muito popular por ser bastante rápida e eficiente.

O que é glFont?

A ferramenta glFont [1] consiste de uma aplicação Windows e uma API para gerar arquivos de fonte e renderizá-los. Foi desenvolvida por Brad Fish (brad.fish@gmail.com), há algum tempo atrás. A versão Symbian C++ da API está neste [link](#).

Limitações

A ferramenta glFont (somente para Windows) é necessária para se construir o arquivos de fonte. A API não é capaz (pelo menos por hora) de ligar com texto Unicode.

Código fonte

Por favor, consulte os artigos ([Matemática de ponto-fixo para Python](#)) e ([Deserializador simples para Python](#)) para o código de matemática e do deserializador de dados.

```

"""
//*****
// glfont2 -- glFont2 API
// Copyright (c) 1998-2002 Brad Fish
// See glfont.html for terms of use
// May 14, 2002
//
// PyS60 port - March 2009
// Luis Valente - lpvalente [at] gmail.com
//
//*****

```

Fontes_baseadas_em_textura_para_OpenGL_ES_(Python)

```
"""

from gles import *
from fixedMath import *      # operações em ponto-fixo
from serial import *        # deserializador

# Estrutura GLFontChar, como armazenada no arquivo
class GLFontCharFile:
    dx = dy = 0      # todos como floats
    tx1 = ty1 = 0
    tx2 = ty2 = 0

# Estrutura GLFontHeaderFile, como armazenada em arquivo
class GLFontHeaderFile:
    tex = 0          # int32
    texWidth = texHeight = 0 # int32
    startChar = endChar = 0 # int32
    chars = 0       # uint32

# classe para a fonte
class GLFont:

    ''' membros "private" '''

    # um caracter simples
    class GLFontChar:
        dx = dy = 0      # todos os tipos como GLfixed
        tx1 = ty1 = 0
        tx2 = ty2 = 0

    # cabeçalho do arquivo de fonte
    class GLFontHeader:
        tex = 0          # GLuint
        texWidth = texHeight = 0 # int
        startChar = endChar = 0 # int
        chars = None     # buffer de memória

        def __init__(self):
            chars = []

    # cabeçalho da fonte
    __header = None

    # vértices, coordenadas de textura e índices (gles.arrays)
    __vertices = None
    __texCoords = None
    __indices = None

    ''' API privada '''
    def loadFile (self, filename):

        # abrir o arquivo
        des = Deserializer (filename)

        # ler o cabeçalho
        self.__header.texWidth = des.readLong()
```

Fontes baseadas em textura para OpenGL_ES_(Python)

```
self.__header.texWidth = des.readLong()
self.__header.texHeight = des.readLong()
self.__header.startChar = des.readLong()
self.__header.endChar = des.readLong()
self.__header.chars = des.readUlong()

# alocar espaço para o array de caracteres
numChars = self.__header.endChar - self.__header.startChar + 1
self.__header.chars = [GLFont.GLFontChar() for i in range(numChars)]

# ler o array de caracteres
for i in range (numChars):
    self.__header.chars [i].dx = float2fixed (des.readFloat () )
    self.__header.chars [i].dy = float2fixed (des.readFloat () )
    self.__header.chars [i].tx1 = float2fixed (des.readFloat () )
    self.__header.chars [i].ty1 = float2fixed (des.readFloat () )
    self.__header.chars [i].tx2 = float2fixed (des.readFloat () )
    self.__header.chars [i].ty2 = float2fixed (des.readFloat () )

# ler os dados da textura com todos os caracteres
numTexBytes = self.__header.texWidth * self.__header.texHeight * 2

texBytes = des.readBytesAsString (numTexBytes)

data = str(texBytes)

# criar textura OpenGL
glBindTexture (GL_TEXTURE_2D, self.__header.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, self.__header.texWidth, self.__header.texHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, texBytes)

# liberar memória usada para a textura
del texBytes

# fechar arquivo
des.close ()
del des

def __destroy (self):
    # apagar o array de caracteres se necessário
    del self.__header.chars [:]
    self.__header.chars = []

''' API "pública" '''

def __init__(self, filename):
    # inicializar o cabeçalho para um estado conhecido
    self.__header = GLFontHeaderFile()

    self.__header.tex = 0
    self.__header.texWidth = 0
```

Fontes_baseadas_em_textura_para_OpenGL_ES_(Python)

```
self.__header.texHeight = 0
self.__header.startChar = 0
self.__header.endChar = 0
self.__header.chars = []

# textura OpenGL
self.__header.tex = glGenTextures (1)

# destruir a fonte anterior, se existir
self.__destroy ()

# carregar arquivo
self.loadFile (filename)

# criar os arrays do OpenGL (gles.array), usados na renderização

v = [0] * 4*2      # 4 vértices 2D
t = [0] * 4*2      # 4 coordenadas de textura 2D
i = [1, 2, 0, 3]  # uma quadrilátero

self.__vertices = array (GL_FIXED, 2, v)
self.__texCoords = array (GL_FIXED, 2, v)
self.__indices = array (GL_UNSIGNED_BYTE, 1, i)

v = None
t = None
i = None

def free (self):
    # destruir a fonte
    self.__destroy()

# apagar a textura
t = [self.__header.tex]
glDeleteTextures (t)

# liberar memória dos arrays
del self.__vertices
del self.__texCoords
del self.__indices

# Determinar os estados OpenGL requeridos para desenhar os caracteres
def beginDraw (self):
    glEnable (GL_BLEND)
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
    glEnable (GL_TEXTURE_2D)
    glEnableClientState (GL_TEXTURE_COORD_ARRAY)

# Desligar os estados OpenGL requeridos
def endDraw (self):
    glDisable (GL_BLEND)
    glDisable (GL_TEXTURE_2D)
    glDisableClientState (GL_TEXTURE_COORD_ARRAY)

# Recuperar as dimensões da textura. Retorna uma tupla com os valores.
def getTexSize (self):
    return (self.__header.texWidth, self.__header.texHeight)

# Recuperar o intervalo de caracteres da fonte, como uma tupla.
```

Fontes_baseadas_em_textura_para_OpenGL_ES_(Python)

```
def getCharInterval (self):
    return (self.__header.startChar, self.__header.endChar)

# Recuperar as dimensões de um caracter, como uma tupla.
def getCharSize (self, character):

    # certificar-se que o caracter está na faixa válida
    character = ord(character)
    if character < self.__header.startChar or character > self.__header.endChar:

        # não é um caracter válido, então não tem nada para retornar
        return (0,0)
    else:

        fontChar = self.__header.chars [character - self.header.startChar]

        # recuperar as dimensões do caracter
        w = fixed2int (fixed_mul (fontChar.dx, int2fixed (self.__header.texWidth) ) )
        h = fixed2int (fixed_mul (fontChar.dy, int2fixed (self.__header.texHeight) ) )

        fontChar = None

        return (w,h)

# Recuperar as dimensões de uma string, como uma tupla
def getStringSize (self, text):

# altura é fixa por hora
    height = fixed2int (fixed_mul (self.__header.chars [self.__header.startChar].dy, int2fixed (self.

# texWidth como ponto-fixa
    texWidthx = int2fixed (self.__header.texWidth)

# calcular o comprimento da string
    widthx = 0

    for c in text:
        # certificar-se que o caracter está na faixa válida
        c = ord(c)
        if c < self.__header.startChar or c > self.__header.endChar:
            continue

        # recuperar o objeto correspondente ao caracter
        fontChar = self.__header.chars [c - self.__header.startChar]

        # recuperar comprimento e largura
        widthx += fixed_mul (fontChar.dx, texWidthx)

        fontChar = None

    # retornar comprimento
    return (fixed2int (widthx), height)

# Desenhar a string. O ponto de referência é o canto superior-esquerdo, e as coordenadas
# são em ponto-fixa
def drawString (self, text, x, y):

    # vertex arrays para renderizar a string
    glVertexPointer (2, GL_FIXED, 0, self.__vertices)
    glTexCoordPointer (2, GL_FIXED, 0, self.__texCoords)
```

Fontes_baseadas_em_textura_para_OpenGL_ES_(Python)

```
# usar a textura de caracteres
glBindTexture (GL_TEXTURE_2D, self.__header.tex)

# iterar sobre os caracteres
for c in text:

    c = ord (c)

    # certificar-se que o caracter está na faixa válida
    if c < self.__header.startChar or c > self.__header.endChar:
        continue

    # recuperar referência para o objeto correspondente ao caracter
    fontChar = self.__header.chars [c - self.__header.startChar]

    # recuperar dimensões
    width = fixed_mul (fontChar.dx, int2fixed (self.__header.texWidth) )
    height = fixed_mul (fontChar.dy, int2fixed (self.__header.texHeight) )

    # especificar coordenadas de textura
    self.__texCoords [0] = fontChar.tx1 ; self.__texCoords [1] = fontChar.ty1
    self.__texCoords [2] = fontChar.tx1 ; self.__texCoords [3] = fontChar.ty2

    self.__texCoords [4] = fontChar.tx2 ; self.__texCoords [5] = fontChar.ty2
    self.__texCoords [6] = fontChar.tx2 ; self.__texCoords [7] = fontChar.ty1

    # e vértices
    self.__vertices [0] = x;          self.__vertices [1] = y
    self.__vertices [2] = x;          self.__vertices [3] = y - height

    self.__vertices [4] = x + width;  self.__vertices [5] = y - height
    self.__vertices [6] = x + width;  self.__vertices [7] = y

    # desenhar
    glDrawElements (GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, self.__indices)

    # ir para o próximo caracter
    x += width
```