2D角色动画快速生成软件

在制作2D动画、游戏时，会大量应用到角色，以及角色动画。制作这些角色和动画需要花费动画师大量的时间、精力。而Internet上存在海量的Gif动画，在不引起版权纠纷的情况下，大部分都可以被用来作为动画片和游戏的非主要角色，或者背景角色。

此外，动画师在进行角色设计的时候，主要关注于角色姿态，制作的角色动画一般都不会发生位移，比如制作走路动作，最终制作完成的角色动画是物体在原地的踏步动作。将角色动画应用于动画片或者游戏当中时，需要小心处理角色位移以及动画播放的关系，否则极易出现角色移动的“漂浮”感。其实，双足或者多足动物在运动过程中至少有一只脚为支撑脚，支撑脚在当前运动帧时不应该发生位移。利用这个原理，本文允许用户交互，为相邻动画帧指定支撑脚，以此来得到角色运动的每帧位移量，从而消除按照固定移动速度所带来的角色“漂浮”感。



Figure 1 原始Gif角色

角色动画

经过多年的研究，计算机图形学在静态成像方面已经相当完善。在当今普通的PC上，我们就能实时制造出令人震撼的效果。但对具有运动物体的场景，怎样使物体以令人信服的方式移动还是一个难题，对需要实时模拟大量运动物体的游戏来说尤其是这样。

从早期的游戏开始，就能够发现角色动画的应用，因为这是玩家对游戏的基本要求。在最初的二维游戏当中，角色被绘制成多张图片（有时也将这些图片拼接为一张大图），这些图片组成角色的运动序列，然后逐张图片绘制到屏幕的不同位置就构成了角色动画。如Figure 2所示，是一个角色在游戏中的各种动作集合，动画序列依次是跳、跑和站立。游戏引擎通过玩家的输入以及角色所处的状态决定在什么位置播放什么动作，以此推动游戏的进行。

1. Jump0.pngJump3.pngJump5.pngJump6.pngJump7.pngJump8.pngJump9.pngJump10.png
2. Run1.pngRun2.pngRun3.pngRun4.pngRun5.pngRun6.pngRun7.pngRun8.pngRun9.pngRun10.pngRun11.pngRun12.pngRun13.pngRun14.pngRun15.pngRun16.pngRun17.png
3. Stand1.pngStand2.pngStand3.pngStand4.png

Figure 2：二维游戏中的角色动画

在id Software的经典作品DOOM里，大群冲向玩家的怪物所带来的紧张感是游戏的卖点之一。在这个时期角色动画实际上利用的还是二维技术，怪物是利用一系列二维的图片来表示的。通过对这些二维图片巧妙的组织和旋转，就可以只利用较少的系统资源实现相当逼真的三维动画效果。随着电脑能力的增强和玩家对游戏要求的提高，新出现的游戏开始采用真正的三维动画模型。

在三维游戏中，动画技术的应用越来越多，小到飘落的树叶，大到咆哮的怪物、奔腾的河流，无不需要使用动画的方式加以表现。为了体现游戏的真实感、艺术性，动画效果一般都是由专门的动画师利用特定的软件提前制作完成，在游戏运行阶段，由游戏引擎对这些动画按照游戏的逻辑、用户输入等条件进行控制，有效地播放出来。当然，也有一些动画效果是通过游戏引擎实时计算得到的，并没有被预先保存。

三维游戏中出现的游戏角色、物品等游戏物体一般都采用三维模型的方式来表现，我们要讨论的角色动画也就是针对三维模型进行变形所得到的动画序列。目前三维模型主要有两种表示方法，一种是网格模型，另外一种是骨骼模型。相应的角色动画类型也分为两种：网格动画和骨骼动画。两种类型的动画都有各自的优缺点，应用于不同场合。网格动画具有更长的历史，计算简单，但需要保存更多的数据，并且和游戏环境交互较少、难于控制。骨骼动画是现在的游戏引擎中常用的动画方式，利用骨骼这一特殊结构来保存角色姿态，只需要很少的数据量就可以表示角色的动画，并且对其控制的灵活度较高，虽然计算复杂，但由于容易和环境交互，能够得到实时计算的姿态，这些优点都使得其在游戏中越来越多地被使用。

然而，不管使用哪种方式，3D模型的制作和应用都是一个难题，有鉴于此，很多游戏以及动画的制作都首先使用2D角色进行测试或者制作原型。特别是目前网络上存在的海量Gif动画图片是一种廉价的角色动画测试资源，将这些资源引入到游戏、动画原型设计当中可以极大提高作品制作效率。

Gif动画文件介绍

GIF(Graphics Interchange Format)的原义是“图像互换格式”，是CompuServe公司在 1987年开发的图像文件格式。GIF文件的数据，是一种基于LZW算法的连续色调的无损压缩格式。其压缩率一般在50%左右，它不属于任何应用程序。目前几乎所有相关软件都支持它，公共领域有大量的软件在使用GIF图像文件。GIF图像文件的数据是经过压缩的，而且是采用了可变长度等压缩算法。GIF格式的另一个特点是其在一个GIF文件中可以存多幅彩色图像，如果把存于一个文件中的多幅图像数据逐幅读出并显示到屏幕上，就可构成一种最简单的动画。

图像互换格式主要是为数据流而设计的一种传输格式，而不是作为文件的存储格式。它具有顺序组织形式而不是随机组织形式。

图像互换格式有五个主要部分以固定顺序出现，所有部分均由一个或多个区块（block）组成。每个块由第一个字节中的标识码或特征码标识。这些部分的顺序为：头块、逻辑屏幕描述块、可选的“全局”色彩表块（调色板）、各图像数据块（或专用的块）以及尾块（结束码）。下面是这些部分的内容：

1. 起头为一个区块，它识别数据流为图像互换格式，并指示如何解释后面的数据所需的最早版本的图像互换格式解码程序（87a或89a）。
2. 逻辑程序描述块定义了：所有后面图像的图像平面的大小、纵横尺寸比以及色彩深度（它类似于产生图像的监视器屏幕）。它还指明后面跟随的是否为“全局”色彩表。
3. 全局色彩表（如果存在）构成一个24位RGB元组的调色板（每种底色为一个字节）。如果后面的像没有其自己的“局部”调色板，那么全局色表就是缺省调色板。
4. 后续数据作为“图形”或“专用”块出现。图形块典型地包含一个或多个位图图像，也可能是覆盖的文本。专用块或者包含一个专用应用程序码，或者包含一句不可打印的注释。
5. 最后的尾块指示值为3B（十六进制）的一个字节，表示数据流已结束。

本软件功能

本软件可以提高动画、游戏的制作效率，提升角色移动的真实感。本软件能完成如下功能：

1. 对Gif动画进行动画帧抽取，可以利用海量互联网动画资源。Figure 1为原始的Gif动画，Figure 3为本软件抽取出的动画帧。
2. 利用支撑脚原理，通过用户交互得到较准确的角色位移信息（Figure 4），从而在动画或者游戏运行时得到真实感较高的动画效果。
3. 角色位移效果可以保存，并进行测试（Figure 5）。
4. 角色位移保存结果以及Gif抽取的动画帧信息可以应用于接下来的动画或者游戏设计中。

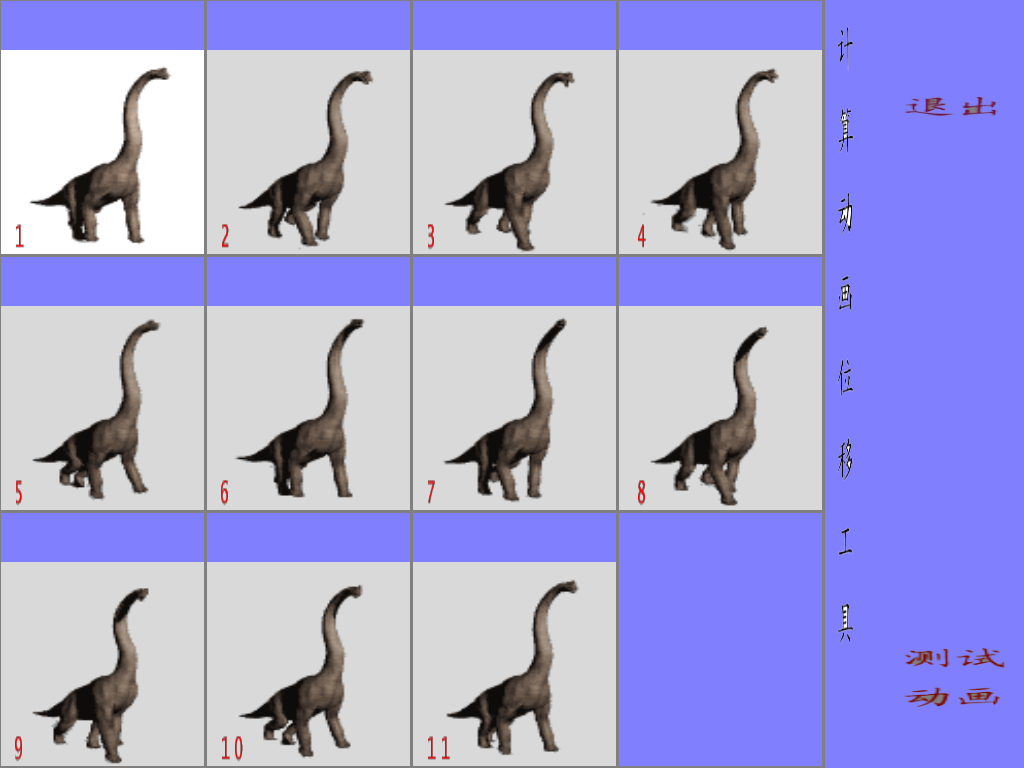


Figure 3 本软件首先将Gif动画帧进行抽取

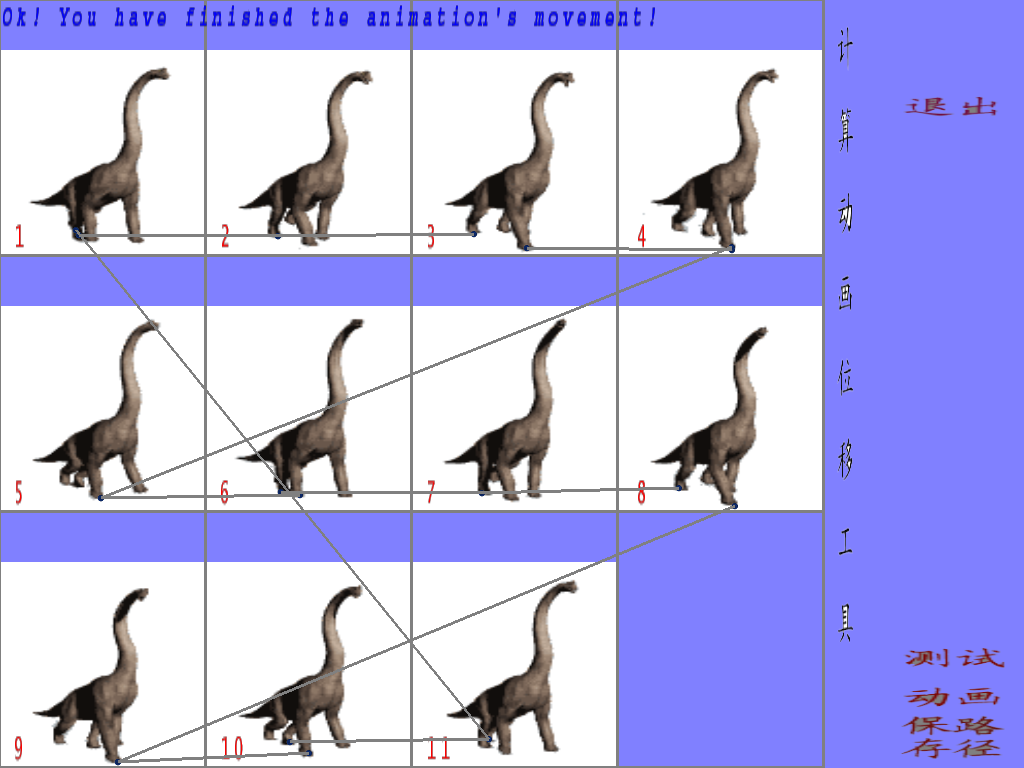


Figure 4 用户交互，找到相邻动画帧对应的支撑脚



Figure 5 利用软件对角色移动进行测试，用户可以交互选定一个平滑曲线路径来控制角色移动

软件框架

本软件在Win32框架下使用OpenGL完成UI界面、图片绘制等操作，主要有以下步骤：

1. **全局变量**

HWND ghWnd ; // 窗口句柄

HINSTANCE ghInstance ; // 程序实例句柄

HDC ghDC ; // GDI 设备环境句柄

HGLRC ghRC ; // 渲染环境句柄

1. **头文件**

#include <windows.h>

#include <GL/gl.h>

#include <GL/glu.h>

需要注意的就是，必须先包含 Windows.h ，然后才能包含 gl.h 和 glu.h 。因为 gl.h 与 glu.h 中可能包含 Windows.h 中定义的宏。

1. **链接库**

此步完全可以通过功能配置来完成，需要包含的库 为 opengl32.lib 何 glu32.lib。

#pragma comment ( lib , "opengl32.lib" )

#pragma comment ( lib , "glu32.lib" )

1. **像素格式 (Pixel Format) 设置**

需要用到的函数的原型：

int ChoosePixelFormat(

HDC hdc, // device context to search for a best pixel format

// match

CONST PIXELFORMATDESCRIPTOR \* ppfd

// pixel format for which a best match is sought

);

BOOL SetPixelFormat(

HDC hdc, // device context whose pixel format the function

// attempts to set

int iPixelFormat,

// pixel format index (one-based)

CONST PIXELFORMATDESCRIPTOR \* ppfd

// pointer to logical pixel format specification

);

这是 Win32 下的 OpenGL 编程必做的事情之一，为 DC 设置像素的格式。

// 设置像素格式

PIXELFORMATDESCRIPTOR pfd ;

int iFormat ;

ghDC = GetDC ( ghWnd );

ZeroMemory ( & pfd , sizeof ( pfd ) );

pfd . nSize = sizeof ( pfd );

pfd . nVersion = 1; // 版本，一般设为

pfd . dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL ;

pfd . iPixelType = PFD\_TYPE\_RGBA ;

pfd . cColorBits = 32;

pfd . iLayerType = PFD\_MAIN\_PLANE ; // 被忽略，为了一致性而包含的

iFormat = ChoosePixelFormat ( ghDC , & pfd ); // 选择一个像素格式

SetPixelFormat ( ghDC , iFormat , & pfd ); // 设置到DC 中

这样的函数完成了像素格式的设置，事实上还可以进行更多的操作，比如设置缓冲区等，下面的代码就是一个设置双重缓冲区的代码。

PIXELFORMATDESCRIPTOR pfd;

ZeroMemory( &pfd, sizeof( pfd ) );

pfd.nSize = sizeof( pfd );

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW | PFD\_SUPPORT\_OPENGL |PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 32;

pfd.cDepthBits = 32;

pfd.iLayerType = PFD\_MAIN\_PLANE;

int iFormat = ChoosePixelFormat( hDC, &pfd );

SetPixelFormat( hDC, iFormat, &pfd );

1. **渲染器环境 (Render Context) 创建**

调用 wglCreateContext 与 wglMakeCurrent 函数 ，这两个函数都是 Windows 下为了兼容 OpenGL 而特别提供的接口，以 wgl 开头。

函数原型：

HGLRC wglCreateContext(

HDC hdc // device context of device that the rendering context

// will be suitable for

);

BOOL wglMakeCurrent(

HDC hdc, // device context of device that OpenGL calls are

// to be drawn on

HGLRC hglrc // OpenGL rendering context to be made the calling

// thread's current rendering context

);

1. **实际绘制和释放资源**

绘制过程和普通的OpenGL程序一直，程序循环调用绘制函数，利用OpenGL的纹理绘制指令完成图片绘制。

程序退出是，首先取消当前的渲染环境选中，然后依次删除渲染环境与设备环境。

需要调用的函数原型：

BOOL wglDeleteContext(

HGLRC hglrc // handle to the OpenGL rendering context to delete

);

// 取消OpenGL ，在程序结束前调用，释放渲染环境，设备环境以及最终窗口句柄。

void DisableOpenGL ()

{

wglMakeCurrent ( NULL , NULL );

wglDeleteContext ( ghRC );

ReleaseDC ( ghWnd , ghDC );

}

软件使用说明

本软件使用了Win32控制台程序结构，使用命令行来传递程序参数。可以使用**bat文件**来进行批处理，需要设定**2个参数**：第一是所要调入**Gif图像的文件名**（如果不再当前文件夹下，需要指定详细路径）；第二是所调入**角色运动的朝向**（r表示向右，l表示向左）。

比如，用户需要处理名为brachio的gif角色动画，则需要按照Figure 6中所示，在bat文件中将必要参数传递进入，其中第二个参数“r”表示，当前的gif角色（如Figure 7所示）为朝右。



Figure 6 利用bat文件来运行程序，并传递gif文件及朝向参数



Figure 7 待处理的角色

程序启动以后，进入**指定连续两帧支撑脚交互指定界面（Figure 4）**，用户交互指定完成以后，点击**“保存路径”**按钮，可以**进入测试动画测试界面（Figure 5）**，用户选择角色移动路径进行测试，如果测试通过，用户可以点击**“保存anm”按钮**，将角色支撑脚信息保存为文本文件。

保存的文件，结合角色动画文件，可以被**应用于2D动画或者游戏**当中。