
日式动画风格非真实感三维实时渲染算法的研究

乐大山 龙晓苑 汪国平

北京大学机器感知与智能教育部重点实验室

北京大学多媒体与人机交互实验室

北京 100871

摘要

本文首先从绘画技法的角度对日式动画风格进行了分析,然后以分析结论为目标设计了一种基于 GPU 着色器的非真实感三维实时渲染算法。使用此算法渲染的 3D 场景所达到的最终效果有以下特点:用黑色等宽的线条勾勒对象,用明暗色块表现光照情况,具有动画风格的阴影。此算法充分利用 GPU 的性能,因此非常适合在游戏、展示软件等实时渲染应用程序中应用。

关键词 日式动画风格 非真实感渲染 图形处理单元 赛璐珞着色法

An Approach to Anime-style Non-photorealistic 3D Rendering

Yue Dashan, Long Xiaoyuan, Wang Guoping

HCI & Multimedia laboratory, Peking University, Beijing 100871

Key Laboratory of Machine Perception (Minister of Education), Peking University, Beijing 100871

Abstract

This article analyzed the drawing styles of Japanese animes, and then presented a Shader-based NPR 3D real-time rendering algorithm that implements these styles. The 3D scenes rendered by this method have the following features: outlining the edges with constant thickness lines, shading the objects with sharp colors and anime-style shadows. This algorithm takes advantages of GPU shaders, and is applicable to real-time rendering applications, such as games and demos.

Keywords: Anime-Style, Non-photorealistic Rendering, GPU, Cel-Shading

1 引言

日式动画的画风较为含蓄、严谨、精致，深受全球观众的喜爱，同时也是各国动画业者竞相模仿的一种风格[8]。使用计算机图形学技术模仿日式动画风格进行实时绘制，可以应用于游戏等人机交互领域，提高产品的艺术感染力和市场竞争力。现已存在不少动画风格渲染的技术，但对于日式动画风格渲染这一特定目标，都存在一些不足。“*Cartoon-Looking Rendering of 3D-Scenes*”[1]是最早论述卡通渲染技术的文章，这篇文章提出的基本框架至今仍未改变，但由于当时技术条件所限，作者采用的描边算法是图像处理技术的方法，并不适于实时渲染。“*Dot3 Cel Shading*”[2]和“*Cel-Shading*”[3]这两篇文章都从实时渲染的角度给出了解决方案，但都属于欧美动画风格，也没有实现阴影。“卡通高光风格化算法及其实现”[9]一文针对动画风格中的高光修正进行了深入研究，但主要针对非实时渲染。在目前发行的商业游戏中，多数卡通渲染均为欧美动画风格，如 Ubisoft 的《XIII》及 Treyarch 的《终极蜘蛛侠》；日本 NAMCO 公司开发的 Xbox 360 游戏《idolm@ster》和《薄雾传说》是日式动画风格游戏的代表，但这两款游戏均无 PC 平台版本，同时其渲染技术也未公开。

本文将首先分析日式动画风格的特征，然后针对这些特征介绍一种借助硬件着色器（Shader）的非真实感实时渲染技术。本文所介绍的技术适合实时绘制，效果可达到日本 NAMCO 公司在其游戏中所实现的那种动画风格渲染技术的水平。

2 日式动画风格的特征分析

这里我所提到的“日式动画”，指的是以日本主流动画片为代表的二维手绘彩色动画片，如《灌篮高手》、《火影忍者》、《名侦探柯南》等。日本著名动画制作人尾泽直志将日式动画风格的特征归纳为线条、色块和人物造型三类[7]，如图 2-1。

线条是日本动画中勾勒对象边界的主要工具。线条宽窄一致，使用同种颜色，仅出现在物体轮廓或者物体的硬折痕处。

日本动画使用色块来表示明暗。所有表面仅被划分为明、暗两种状态，之间有明确的明暗分界线。在日本动画中，暗色块是由背向光源的区域和被其它物体遮挡产生的阴影共同构成的。在头发或金属物体表面上有时也会存在高光。

日式动画人物的造型化充分体现了日本设计师对唯美的执着。动画人物的比例都比较理想化，手脚修长，面部圆润、皮肤光滑，大眼睛、小鼻子。美少女造型身高可达到九个头，腿长经常会达到身高的 1/2。服装与饰物款式时尚，但色彩简单，多数服饰都是单色或大色块。绝大多数表面都是光滑圆润的，极少使用锐利的边角。

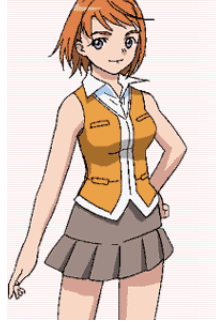


图 2-1 日本动画风格特征：单色等宽线条勾边，明暗用色块表示，明暗分界线产生自照射角度（明暗）与对象遮挡（阴影）

3 日式动画风格三维渲染技术的原理

日式动画风格的三个特征中，除了人物造型需要由 3D 模型师来实现外，另外两种特征需要通过渲染算法来实现。针对单色等宽线条这一特征，我选择并改进了背面线框描边算法[2]；由于色块特征中的暗色块是由明暗和阴影两方面构成的，因此我综合了传统的 Cel-shading 明暗算法[2]和基于深度阴影图的纹理阴影技术[4]，引入阴影技术并将两种暗色调进行混合是我的创新之处。

3.1 描边算法的原理

我们需要为模型绘制单色等宽线条，将模型的边线勾勒出来。寻找边线看起来是一个很困难的问题，因为在我们的渲染流程中没有一个可以直接遍历所有边，并且同时了解每条边两侧表面法向量的步骤。虽然前人有利用图像处理技术来寻找边界的例子，但这种基于图像空间的方法会大幅增加运算量，而且不能在 GPU 中运行。

边线最大的特征就是，与它邻接的面一个向前、一个向后。恰好显示卡硬件提供正面剔除和背面剔除的选项。在渲染时，我们在不同的遍中分别渲染正面和背面，利用他们的差异，就可以将边界绘制出来。

3.2 明暗色块光照模型

Cel-shading 明暗算法是以 Phong 局部光照模型为基础并加以改进得到的。在传统的 Phong 局部光照模型中，反射光强度可以用以下算式表示：

$$I = k_a I_a + I_i (k_d (\mathbf{L} \cdot \mathbf{N}) + k_s (\mathbf{R} \cdot \mathbf{V})^n)$$

可见，对于固定位置的光源和确定的材质，反射光强度是随面法向量和观察方向变化的连续函数[6]。通过将漫反射光、镜面反射光两项离散化，我们就可以得到色块化的效果。

3.3 基于深度阴影图的纹理阴影技术

基于深度阴影图的纹理阴影技术最早在“*Casting Curved Shadows on Curved Surface*” [4] 文章中提出。其基本思想是，如果我们从光源位置观察场景，那么我们能看到的区域就是光源照亮的区域，看不到的区域就是阴影区域。具体过程可描述如下：

1. 首先我们要创建一个纹理，这个纹理将用来保存场景深度信息，并称它为“深度阴影图”。
2. 假设在光源处另有一个摄像机，它从光源的位置观察场景。我们从这个光源摄像机的角度绘制场景，将深度信息将做为每个像素的“颜色”绘制到“深度阴影图”纹理中。这些深度信息反映了这条直线上离光源最近的对象与光源的距离。如图 3-1 (b)。
3. 回到正常摄像机视角。对于每个需要接收阴影的材质，我们将刚才生成的“深度阴影图”纹理映射到其上。对于每个像素，我们将其空间坐标变换回上一步设置的光源摄像机空间，此时深度信息也表示了该点与光源的距离。我们对每个像素进行深度测试，如果该像素的深度大于“深度阴影图”纹理中对应像素中保存的深度，则说明该像素处于阴影中，否则该像素处于阴影外。图 3-1 (a)为测试结果。

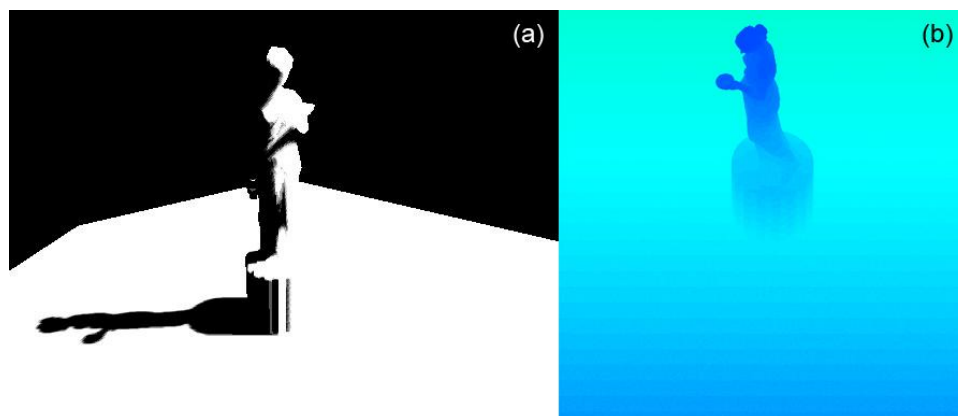


图 3-1 深度阴影图算法示意。(a)投射出的阴影 (b)从光源摄像机生成的深度阴影图纹理[5]

4 日式动画风格三维渲染技术的实现

我将实现动画风格三维渲染技术分为以下几个步骤：

- 准备模型：创建符合日式动画风格的三维模型及纹理；
- 描边：用单色等宽的线条描出模型的边界和硬边；
- 光照渲染：用有明显界线的明暗色块来渲染模型；
- 阴影渲染：借助深度阴影图技术（Depth Shadow Map）实现阴影，并与光照渲染的结果加以混合。

下面我将按照这四个步骤的顺序逐一加以说明。

4.1 准备模型

严格地说，准备模型应是三维美工师的工作，并不应属于渲染技术所讨论的范畴，然而模型作为整个渲染算法的输入，其质量的好坏却在很大程度上影响着渲染的最终效果。

首先，用于动画风格渲染的模型要求表面尽量光滑，建模时应首选使用 NURBS 曲面，在建模完成后再转换导出。其次，动画人物的表情复杂多变，往往不符合上述动画风格的特征，所以面部五官不要使用模型的凹凸来描述，而应该使用纹理贴图。最后，除了面部纹理外，其它纹理都应该只包含色块，其颜色是该区域的基色。所谓基色，就是材质在完全明亮但无高光的环境中的颜色，也就是未作明暗处理的颜色。另外还应避免使用纯白色或者纯黑色，这样会使明暗处理后缺少层次感。一般白色偏蓝，黑色偏棕[7]。

4.2 描边

DirectX 和 OpenGL 都提供了线框模式（wire-frame）选项。在这种模式下，计算机不绘制模型的表面，只使用等宽线条绘制多边形的边。结合线框模式和硬件剔除功能，我们可以较容易地实现一种符合日式动画风格特征的描边算法。

背面线框算法一共 2 遍，第一遍正常绘制，第二遍进行描边。在第二遍绘制时我们进行如下设置。图 4-1 形象地描述出了算法过程。

1. 启用线框模式；
2. 启用硬件剔除功能，剔除面向摄像机的表面，只保留背向摄像机的表面；

3. 将所有顶点的颜色设置为黑色或指定的描边颜色；
4. 设置线宽为 2 像素或以上；
5. 设置深度偏移量为+1。

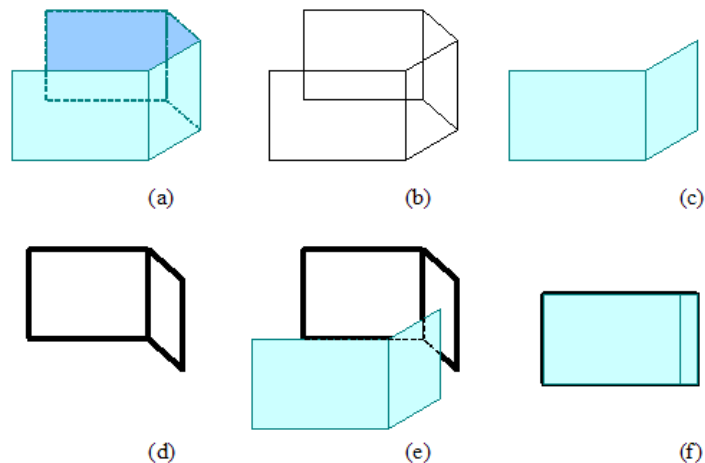


图 4-1 背面线框算法示意

- (a) 原始模型 (b) 线框模式绘制 (c) 第一遍正常绘制模型（硬件背面剔除）
 (d) 线框模式、正面剔除、2 像素线宽、深度偏移量+1 的模式绘制模型
 (e) 两遍绘制的结果 (f) 从摄像机角度看，模型有了边线

在默认情况下，线框模式是以 1 像素宽度的细线进行绘制的，但由于边线和与它邻接的正表面的边是重合的，实际上只有线框宽度的 1/2 会被显示出来，因此我们须将线宽设置为 2 像素或以上。DirectX 中不能直接设置线宽，可通过启用硬件反混淆（anti-aliasing）或自行实现宽线条来抑制线条不连续的问题。设置深度偏移量可以保证在深度缓冲区中线框均比正面更“深”一些。这样可防止模型距离摄像机较远时发生深度冲突（Z-fighting）。

这一算法能够有效地找到边线，对于向内折的硬折痕，也可以勾画出来。勾边线条粗细一致，且不会因对象的远近而发生变化。因此，本算法满足了我们前面提出的需求。

4.3 光照渲染

光照渲染部分是整个动画风格渲染技术的核心。它用具有明显分界线的色块来抽象地反映光照情况，使模型具有平面绘画的色彩效果。

反射光强度连续变化是造成光滑表面色彩连续变化的原因，因此我们要将其进行二值离散化处理。我们可以利用纹理映射机制实现从连续值到二值的映射。首先我们建立了一个 16 像素宽的一维纹理，我们称其为“漫反射光照纹理”，如图 4-2。



图 4-2 漫反射光照纹理

实现明暗色块光照的关键就在这里：在对每像素进行纹理映射时，我们将每像素的反射光强度数值当作上述纹理的纹理坐标。通过图素查询，我们就实现了反射光强度的二值化。将这个值与该像素的原始色彩混合，就得到了基色与暗色两个色块，如图 4-3。

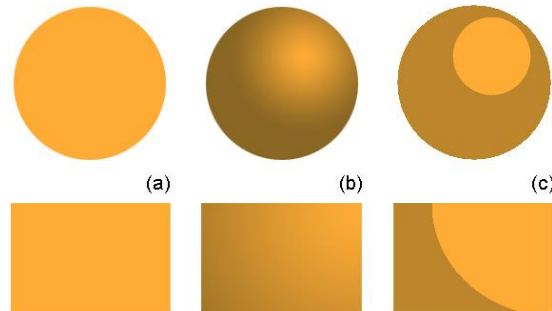


图 4-3 明暗色块光照模型示例。

(a)只使用基色渲染，(b)使用传统光照模型渲染，(c)使用明暗色块光照模型渲染。

对于存在高光的光泽表面，高光的处理方法与阴影基本相同。不同的是阴影色是将基色变暗，而高光则是将基色变亮。另外，由于高光面积应该较小，其二值化的阈值也应更高。

现在我们来实现明暗色块光照模型。开始绘制前，我们要将原始颜色（基色）纹理、漫反射光照纹理和高光纹理载入，分别作为第 0~2 层纹理。模型的顶点数据中应该已经包含了基色纹理（第 0 层）的纹理坐标信息。在对每顶点处理时，我们会计算出该点的漫反射强度与镜面反射强度，用它们作为第 1 层和第 2 层纹理的坐标。我们将分别使用 Vertex Shader 和 Pixel Shader 来实现上述每顶点和每像素的操作。

程序清单 4-1 给出了一个使用 Cg 语言实现的明暗光照算法的像素着色器部分。

程序清单 4-1 明暗光照算法的像素着色器

```
void main_fp(
    float2 uv          : TEXCOORD0,
    float diffuse      : TEXCOORD1, // 在顶点着色器中求得的漫反射因子
    float specular     : TEXCOORD2, // 在顶点着色器中求得的镜面反射因子
    uniform sampler2D myTexture : register(s0), // 原始纹理
    uniform sampler1D diffuseRamp : register(s1), // 漫反射光照纹理
    uniform sampler1D specularRamp : register(s2), // 高光纹理
    uniform float darken, // 阴影色加深系数
    uniform float lighten, // 高光增亮系数
    out float4 colour : COLOR
)
{
    // 通过对“漫反射光照纹理”进行纹理查询，得到该像素的明暗值，并按系数进行放缩
    float darkness = tex1D(diffuseRamp, diffuse).x;
    float final = (darkness - 1) * darken;
    // 用相同的方法处理高光
    final += tex1D(specularRamp, specular).x * lighten;
    // 对 RGB 三个通道分别将最终调整值 (final 变量) 与该像素的原始纹理进行加色混合
```

```
    colour = float4(tex2D(myTexture, uv).xyz + float3(final), 1);  
}
```

4.4 阴影渲染

首先我们需要创建一个新纹理，它用来保存深度信息，尺寸为 $2^n \times 2^n$ ，例如 1024×1024 。

为了获得“深度阴影图”，我们需要为绘制算法增加 1 遍。这一遍从光源角度进行绘制，任务是将光源摄像机空间中的深度信息绘制到“深度阴影图”纹理中。算法可以表述为：

1. 对于每顶点：
 - a) 将顶点的空间坐标映射到光源摄像机坐标系；
 - b) 将顶点的深度信息以纹理坐标的形式保存到顶点数据结构中；
2. 对于每像素：将纹理坐标（即该点深度）当作“颜色”绘制到“深度阴影图”纹理中。

在绘制阴影接收对象时，我们要将“深度阴影图”纹理坐标映射到对象的每个顶点上，然后我们对于每个像素进行一次深度测试。通过测试的像素正常渲染，测试失败的像素则被标为阴影。我们可以使用 PCF（Percentage Closer Filtering）模板过滤器[5]（图 4-4）来减少阴影纹理中的“锯齿”。日式动画中阴影与明暗中的暗色调是处于同一色块中的，因此在混合时二者只取其一，如程序清单 4-2。

$$\frac{1}{5} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

图 4-4 5 点 PCF 模板过滤器的矩阵表示

程序清单 4-2 阴影与暗色调混合的代码片断

```
// 此代码片断位于像素着色器中  
// 对于每像素，如果它同时处于暗色调和阴影之中，则只选择其中颜色较深的值绘制  
float darkness = (shadow > diffuse) ? diffuse : shadow;  
// 存在阴影的地方，不能产生高光  
float brightness = (shadow > 0.0f) ? tex1D(specularRamp, specular).x : 0;  
// 继续明暗色块光照渲染...
```

5 实验结果及分析

5.1 渲染效果对比

为了测试上述算法的效果与性能，我利用 C++ 和开源渲染引擎 OGRE 实现了此算法并制作了一个展示程序（DEMO）。图 5-1 和图 5-2 分别对描边效果和明暗效果进行了对比。



图 5-1 描边算法效果比较

(a)本文介绍的背面线框算法能够连贯等宽地勾边 (b)现在较流行的扩展模型算法勾边不够连贯
(c)传统 Cel-shading 描边算法不能有效勾边 (d)无描边

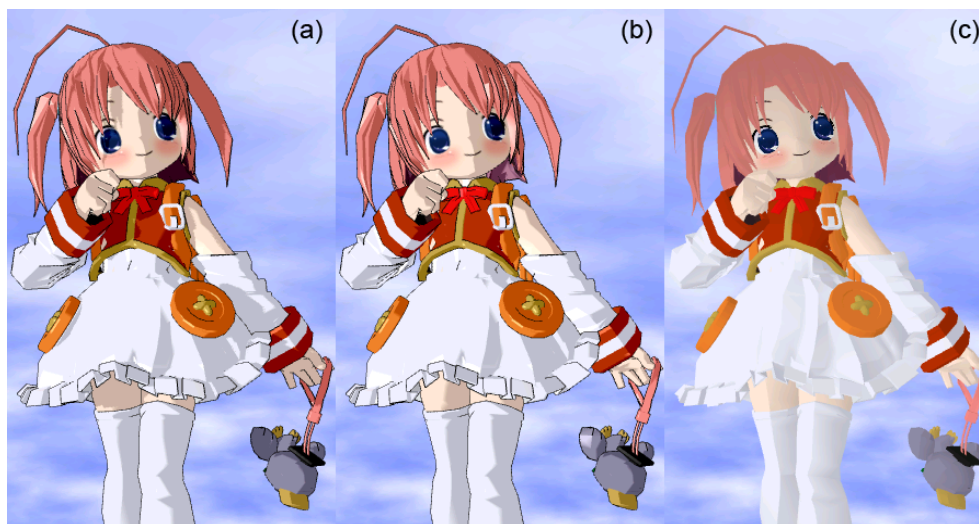


图 5-2 明暗算法效果比较

(a)本文引入了阴影技术并与明暗效果混合，使人物在侧光下更生动
(b)传统 Cel-shading 明暗算法不支持阴影 (c)普通渲染

5.2 性能与优化

该 DEMO 中的模型共有约 56000 个多边形，在启用阴影、4 倍硬件反混淆、1024x768 分辨率、色深 32 位的情况下进行测试，帧速率如表 5-1。

表 5-1 日式动画风格非真实感渲染 DEMO 性能比较

编号	CPU	内存	显卡	平均帧速率
----	-----	----	----	-------

A	AMD Athlon64 3000+	1.0GB	NVIDIA GeForce 8500GT	136 fps
B	Intel Core Duo 1.8GHz	2.0GB	NVIDIA GeForce 8400M	84 fps
C	Intel P4 1.86GHz	1.5GB	ATI MOBILITY X300	46 fps
D	Intel P4 3.0GHz	1.0GB	NVIDIA GeForce 5200FX	13 fps
E	Intel P4 3.0GHz	1.0GB	Intel 82915G	5 fps
F	Intel P4 1.7GHz	512MB	ATI 7500	(无法运行)

由于本解决方案中大量的渲染任务都是在 GPU 上执行的，所以其渲染性能主要取决于显卡，与 CPU 及内存容量并无明显的相关性。本解决方案主要是使用基于 VS2.0/PS2.0 规格的着色器实现，理论上说要求 NVIDIA GeForce 6200 或 ATI Radeon 9000 以上级别的显卡才可以正确运行，因此在低于此标准的显卡上会出现速度缓慢甚至无法执行的情况。而对于较新的显示硬件，其渲染性能有大幅提升。可见，随着高性能的显示硬件不断普及，此解决方案是可以被广泛使用的。

5.3 已知问题与替代性措施

本解决方案仍然存在一些缺陷。在表 5-2 中我列举了几项，并尝试给出替代性措施以减少这些缺陷带来的影响。

表 5-2 日式动画风格非真实感渲染技术的部分缺陷及其替代性措施

现存缺陷	替代性措施
对早期显示硬件的支持程度不佳。在不支持着色器的显卡上甚至不能执行。	<ul style="list-style-type: none"> ● 为应用程序设置“最低配置”要求； ● 同时设计一个固定功能算法，当此解决方案不被支持时自动选择此算法。
实现阴影所采用的纹理阴影（Texture Shadow）技术有一定局限性，它对聚光灯光源、平行光光源都支持较好，但对点光源却支持不佳；	<ul style="list-style-type: none"> ● 避免使用点光源，尽量选用聚光灯光源，并尽量保证光源恰好照射到要进动画风格渲染的物体上； ● 根据场景特点选择适合的“光源摄像机”投影变换设置。
实现阴影所采用的纹理阴影（Texture Shadow）技术受“阴影纹理”大小的影响，在某些情况下会出现锯齿状混淆。	<ul style="list-style-type: none"> ● 选用更大尺寸的纹理； ● 避免摄像机从过于极端的角度观察接收阴影的对象。

6 结论

本文首先对日式动画风格的技术特征进行了分析,并以此提出了实现动画风格渲染的技术目标。接下来,本文逐一介绍了明暗渲染、描边和阴影渲染这三个步骤的原理,并给出了一些实现细节。引入阴影技术并与现有的明暗渲染效果加以混合是本文的创新点。最后,本文也对此本解决方案的性能和缺陷进行了总结。

本文介绍的日式动画风格非真实感实时渲染技术,有针对性地模拟了日式动画片的技法风格,不仅能产生艺术化的效果,而且可以较好地利用显卡硬件提供的特性,将 CPU 时间节约出来留作他用。本技术非常适合在游戏、展示软件等实时渲染应用程序中应用。

参考文献

1. Philippe Decaudin, “*Cartoon-Looking Rendering of 3D-Scenes*”, Research Report INRIA #2919, June 1996
2. Ron Barbosa, “*Dot3 Cel Shading*”, ShaderX3: Advanced Rendering with DirectX and OpenGL, November 2004
3. Sami “MENTAL” Hamlaoui, “*Cel-Shading*”,
<http://www.gamedev.net/reference/programming/features/celshading/>
4. L. Williams, “*Casting Curved Shadows on Curved Surface*”, Computer Graphics 12,3, pp. 270-274, August 1978
5. Anirudh.S Shastry, “*Soft-Edged Shadows*”,
<http://www.gamedev.net/reference/articles/article2193.asp>
6. [美]Alan Watt 著,包宏译,《3D 计算机图形学(原书第 3 版)》,机械工业出版社,2005
7. [日]尾泽直志著,谢时新、杨雁群译,《日本动漫人物造型基础教程》,上海人民美术出版社,2006
8. Japanese Economy Division, “*Japan Animation Industry Trend*”, JETRO Japan Economic Monthly, June 2005, http://www.jetro.go.jp/en/market/report/pdf/2005_35_r.pdf
9. 苏延辉,韦欢,费广正,石民勇,“卡通高光的风格化算法及其实现”,《中国计算机图形学进展 2006》,2006 年 6 月