

**基于物体表面结构的**

**材质纹理编辑系统的设计与实现**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术（软件技术）**

**学生姓名 欧阳耀斌**

**学生学号 200830580288**

**目 录**

**[摘 要](#_Toc326519732)** [I](#_Toc326519732)

**[Abstract](#_Toc326519733)** [II](#_Toc326519733)

**[第一章 引言](#_Toc326519734)** [1](#_Toc326519734)

[1.1 材质和纹理 1](#_Toc326519735)

[1.2 研究现状 1](#_Toc326519736)

[1.3 基于物体表面结构的材质纹理编辑系统 2](#_Toc326519737)

[1.4 拟解决的关键问题 3](#_Toc326519738)

[1.4.1物体表面上的纹理绘画 3](#_Toc326519739)

[1.4.2无缝贴图与纹理过滤的实现 3](#_Toc326519740)

[1.5 系统功能要求 3](#_Toc326519741)

[1.6 本文安排 4](#_Toc326519742)

**[第二章 纹理画笔的设计与实现](#_Toc326519743)** [5](#_Toc326519743)

[2.1 纹理画笔的作用 5](#_Toc326519744)

[2.2 相关算法研究 5](#_Toc326519745)

[2.3 纹理画笔的实现 6](#_Toc326519746)

[2.3.1插值 6](#_Toc326519747)

[2.3.2 画笔定位 8](#_Toc326519748)

[2.3.3像素映射 8](#_Toc326519749)

[2.3.4像素更新 11](#_Toc326519750)

[2.4 结果与分析 12](#_Toc326519751)

[2.5 本章小结 13](#_Toc326519752)

**[第三章 纹理接缝消除算法的设计与实现](#_Toc326519753)** [14](#_Toc326519753)

[3.1 纹理接缝的产生原因 14](#_Toc326519754)

[3.2 现有算法研究 15](#_Toc326519755)

[3.3 接缝填充算法具体实现 16](#_Toc326519756)

[3.4 结果和分析 18](#_Toc326519757)

[3.5 本章小结 20](#_Toc326519758)

**[第四章 系统实现与测试](#_Toc326519759)** [21](#_Toc326519759)

[4.1 系统总体架构设计 21](#_Toc326519760)

[4.1.1 界面操作层 21](#_Toc326519761)

[4.1.2场景管理层 22](#_Toc326519762)

[4.1.3底层操作层 23](#_Toc326519763)

[4.1.4 主要的类及其功能 23](#_Toc326519764)

[4.2 系统环境选择 25](#_Toc326519765)

[4.2.1 Qt概述 25](#_Toc326519766)

[4.2.2 Qt在系统中的运用 27](#_Toc326519767)

[4.3功能模块描述 28](#_Toc326519768)

[4.3.1 文件操作 29](#_Toc326519769)

[4.3.2 视图操作 29](#_Toc326519770)

[4.3.3 颜料控制 30](#_Toc326519771)

[4.3.4 画笔控制 33](#_Toc326519772)

[4.3.5 物体变换 35](#_Toc326519773)

[4.3.6 历史记录 36](#_Toc326519774)

[4.3.7 材质层 36](#_Toc326519775)

[4.4系统功能测试 37](#_Toc326519776)

[4.5 本章小结 40](#_Toc326519777)

**[第五章 结论](#_Toc326519778)** [41](#_Toc326519778)

**[参考文献](#_Toc326519779)** [42](#_Toc326519779)

**[致谢](#_Toc326519780)** [44](#_Toc326519780)

摘 要

在三维图形学领域，材质和纹理是表现物体表面细节的重要手段。但是，在当前主流三维软件的材质编辑系统中，不同属性的纹理通道之间往往是不相关联的。例如，对于破旧的物体，凹陷部分由于灰尘的堆积，往往颜色比较深。可是在现有的编辑系统中，很难自动捕获凹陷部分与其余部分的颜色区别，更无法自动对凹陷部分绘制更深的颜色。换而言之，颜色通道很难跟其他通道有效地关联。在实际的制作中，美工人员往往先用photoshop，Zbrush等软件绘制颜色贴图，然后设法把颜色贴图修改为对应通道的灰度图，同时利用Zbrush生成法线贴图，最后把所有的贴图导入3ds max、Maya这些软件的材质编辑系统。这种办法的缺陷是事先很难把握最终的效果。创作人员需要具有一定的经验，创作的过程中需要反复调整和渲染，才能做出比较好的效果。

针对这种情况，本文提出了一种新的材质纹理编辑方式。具体来说，就是直接像刷油漆那样往物体表面绘制材质和纹理，同时仿照photoshop软件中的图层操作，引进了材质层概念，使得各层材质具有厚度。绘制材质的时候，根据表面厚度等信息，可以自动调整绘制的颜色。这样就可以较好地模拟物体表面的细节。

本文首先对当前的材质编辑系统进行系统地介绍和分析，指出现有系统的弱点，然后提出一种新的材质编辑方式，并详细地介绍了其采用的一些算法，并进一步实现了该材质编辑系统，最后本文还对这种材质编辑方式的实际结果进行了分析说明。

根据实验结果分析，本系统初步具备了材质编辑功能，对于某些模型表面，能够实现比较满意的渲染效果。但由于时间所限，系统功能尚不完备，相关算法的效率有待提高，因此尚有较大的改进空间。

**关键词**：三维图形学；材质编辑；纹理贴图

Abstract

In 3D computer graphics, materials and textures are important to show surface details. However, texture channels of different material properties are often not associated with each other in the material editors of current 3D computer graphics software. For example, due to the accumulation of dust, the color is usually quite deep in concave parts of obsolete things. But it’s quite hard to distinguish concave parts from others in such obsolete things in current material editors, not to mention rendering a deeper color in these concave parts automatically. In short color channels are difficult to be linked with other channels in current material editors. In fact artists are required to use photo-editing software such as Photoshop and ZBrush to draw the color map firstly, then change it to the grayscale one for the corresponding channel. Meanwhile a normal map is created, using the software ZBrush. At last all maps are imported into the material editor of 3ds max or Maya.  Because of many human factors in the manufacturing process, it’s quite possible to get a bad result in this approach. In order to get ideal results, there must be repeated adjustments and renderings.

In order to solve this problem, we tried a new approach in this thesis to edit materials and textures, just like painting directly on object surfaces. Inspired by the concept of photo layers in Photoshop, we created the concept of material layers, which have a thickness property. When painting on surface, the painted color will be adjusted automatically according to surface thickness properties, so that the details of object surfaces can be simulated successfully.

In this thesis, mainstream material editors were firstly introduced and analyzed. Then we presented the above new material-editing approach, including the detailed descriptions of related algorithms and the specific implementation of the whole material editor. Finally, the experiment results were shown and analyzed.

As the results showed, our material editor has basic material editing features. Some types of surface rendering can be simulated perfectly. However, due to time constraints, some features are still not completed yet and some algorithms are required to be improved for high efficiency.

**Keyword**: 3D computer graphics, Material Editing, Texture Map

第一章 引言

1.1 材质和纹理

在计算机图形学中，用材质属性来表征物体的质地。材质可以看成是材料和[质感](http://baike.baidu.com/view/132777.htm" \t "_blank)的结合。它是表面各可视属性的结合，这些可视属性是指表面的色彩、[纹理](http://baike.baidu.com/view/49346.htm" \t "_blank)、光滑度、透明度、反射率、[折射率](http://baike.baidu.com/view/42992.htm" \t "_blank)、发光度等。图形程序根据这些属性，按照一定的规则，可以计算出物体表面的颜色值，从而得到具有真实感的图像。

纹理则用于表现物体表面的质感细节。在现实世界中的物体，其表面通常有它的表面细节，即各种纹理，如刨光的木材表面上有木纹、建筑物墙壁上有装饰图案、机器外壳表面有文字说明它的名称、型号等，如果忽略这些细节，表面就会显得过于光滑和单调，看起来反而不真实。具体来说，在图形学中，纹理主要有两种：一种是通过颜色色彩或明暗度变化体现出来的表面细节，这种纹理称为颜色纹理。另一类纹理则是由于不规则的细小凹凸造成的，例如桔子皮表面的皱纹，这类纹理称为凹凸纹理。

1.2 研究现状

当前的材质纹理编辑系统，如3ds max的材质编辑器，maya的hypershade编辑器，其编辑流程大体上遵循以下步骤：先往物体赋予一个材质，然后再往材质添加各种纹理贴图，以控制材质的各种属性，如物体表面的颜色、凹凸、粗糙度等等。此方法对于表面结构比较简单的物体，例如一个崭新的、表面基本没有瑕疵的木桌子，能够比较快捷地表现出物体的材质质感；但对于表面结构比较复杂的物体，例如破旧的墙壁，较难快速地达到满意的效果。在实际编辑时，往往需要预先在photoshop绘制好各个材质属性对应的纹理，然后回到三维软件，逐个指定这些纹理。这种编辑方式有几个弊端：首先是不直观，无法即时在三维软件看到编辑的结果。其次，因为有时候一些表面细节会同时牵涉到几个纹理，例如，为了表现粗糙的墙面上新刷的油漆，除了需要在颜色贴图中画出油漆的颜色之外，还要在光泽度贴图中把油漆区域的光泽度调高（因为油漆未干时表面反光比较强）。如果要表现油漆顺着墙流下来的现象，还要修改一下凹凸贴图，以表现液滴末端由于油漆积聚鼓起的现象。

针对上述弊端，当前已经有一些软件，例如bodypaint，能够直接在物体上绘制纹理；而3d coat甚至可以通过控制画笔属性同时修改颜色、高光、凹凸这几个通道。可是，bodypaint仍然是基于传统的先赋予材质再加贴图的方式，而且编辑，显示精度不足。3d coat 可以设置画笔同时影响表面的颜色，高光，凹凸等属性，可以说朝着新的编辑方式迈出了一步，可惜其画笔工具比较粗糙，表现细腻的表面稍微显得力不从心。

1.3 基于物体表面结构的材质纹理编辑系统

针对上述研究现状，本文拟提出一种新的材质纹理编辑方式。具体来说，即抛弃传统的先给材质，再画纹理的方式，直接往物体表面绘制材质和纹理。同时仿照photoshop图层，引进材质层概念，使各层材质具有厚度，然后，模拟出一些真实世界的工具，例如刮刀可以清除一定深度的多层材质，抹布可以提高表面的光泽度， 而画笔则可以把一定属性的材质（而不是颜色）像涂油漆那样涂到物体表面。而且，画笔可以根据表面的一些信息，模拟出一些效果。例如根据各层材质叠加的总高度，模拟出表面凹槽积聚灰尘、污渍的效果，或者少量油漆涂在凹凸不平的表面的效果。另外，实现读取photoshop的abr格式笔刷文件，利用网上丰富的笔刷资源，增强画笔的表现力。在编辑完成后，用户能够根据表面的各种信息导出各个通道的贴图，而对于材质属性，则可打包成一个mat文件整体导入3ds max。系统具备了上述功能，就可以改变现有的材质编辑流程。

a) 原图 b) 灰度图



图1-1 斑驳的墙壁

例如对于图1-1a中斑驳的墙壁，可以看出它大体由以下三层组成：最底层是砖层，接着是水泥层，最上面是淡黄色的墙面。通常情况下，这样的效果可以在photoshop上使用多个图层来实现，可是对于三维模型，表面的凹凸程度也是一个很重要的细节，而控制凹凸程度的高度图很多时候是不能简单通过修改颜色贴图来实现的。例如,如果把图1-1a转成灰度图，窗框阴影的地方（黄色框内部分）就会比较暗，而在高度图中这就意味着这些地方会比较低，这是不符合事实的。因此，美术人员在创作此类贴图时往往需要进行细致的修改，或者是通过Zbrush等雕刻软件先把表面的凹凸细节描绘出来，再在上面画贴图，但这种方式同样把颜色和表面凹凸的产生过程分离了，显得不够直观。

如果根据墙的结构定义三个材质层，即砖头层，水泥层和表面层，编辑的时候同时修改各个图层的颜色和厚度，最后根据这些信息生成对应的纹理贴图，整个工作流程就显得比较自然了。

1.4 拟解决的关键问题

以下是本系统拟解决的一些关键问题。

1.4.1物体表面上的纹理绘画

简单来说，此功能是二维绘画软件中画笔工具的三维版本。利用这个工具，用户能够在物体表面直接绘制纹理。

由于是在三维空间中绘画，因此相比二维画笔有一些技术上的难题需要解决。最关键的一点，是如何把笔刷的像素映射到物体的纹理空间中。其次，如何把光标在屏幕的位置映射到三维空间中的位置，也是一个要解决的问题。

在当前的实现中，较好地解决了上述问题。关于系统具体采用的算法，会在第二章进行详细的介绍。

1.4.2无缝贴图与纹理过滤的实现

在系统最初的设计中，没有考虑到纹理接缝的问题，导致某些情况下物体表面会出现明显的裂纹。于是在随后的实现中，对此问题进行了比较详细的研究，并找到了一个减小接缝处瑕疵的方案。在本文第三章，会对此方法做比较详细的介绍。

1.5 系统功能要求

这个系统的目标是能够提供一种新的，而且是自然的、容易操作的材质编辑方法。为了实现这种材质编辑方法，系统必须具备以下功能：

1. **模型读入**。这是其他所有功能的前提。系统必须支持至少一种模型格式。在当前的实现中，系统可以比较稳定地读入obj格式。
2. **模型的选择、平移、旋转、缩放功能**。这是一个基础功能，只有具备了这个功能，用户才能方便地对模型进行控制。
3. **视角的控制**。这也是一个基础功能。
4. **画笔**。这是整个软件最核心的功能。其中涉及到画笔的正确定位，颜料从笔刷空间到物体uv空间的映射，以及纹理过滤等等方面。过程比较复杂，对效率要求很高。而且，为了做出丰富的效果，需要提供多种控制方式。
5. **材质层**。这也是整个软件比较关键的一个功能。其中，选择合适的材质层的混合方式，对最终效果有明显的影响。而材质层的更新效率，则影响到绘制能否流畅进行，是实现的难点。

当然，如果希望真正把整个材质编辑过程都移到三维空间中进行，则需要更多的工具。例如如果要仿照photoshop的路径工具，最自然的办法就是做一个能够附着在模型表面的路径工具，可是实现起来需要解决很多问题。又例如photoshop的选区工具，如果在三维上实现，就会涉及到填充，裁剪或者采样过滤等等算法，这些算法即使能够实现，也是需要花比较多的精力，如果加入羽化功能，那就更难了。另一方面，在物体表面上直接绘画，需要先对网格模型展开纹理坐标。当前这方面已经有相关的算法，可是考虑到时间问题，暂时不实现展开功能，而是假定网格模型已经展开了纹理坐标。

综上所述，系统需要提供最基本的材质编辑功能，并尽量使得这些功能能够实现丰富的效果。对于一些不是必要的，而且实现比较复杂的功能，暂时不考虑实现。

1.6 本文安排

以上简要介绍了材质和纹理的基本概念、系统的设计目标以及要解决的问题。在第二章和第三章，将介绍系统中采用的两个关键算法——纹理画笔绘制算法以及接缝消除算法。在第四章，将对系统的实现进行详细的介绍，同时给出测试结果。在结论部分，将对当前系统的优缺点进行评估，并指出今后的研究方向。

第二章 纹理画笔的设计与实现

纹理画笔是本系统的一个核心功能。本章将对此做详细的介绍。

2.1 纹理画笔的作用

本系统的设计目标是使用户能够通过直接在模型上绘制纹理。而要提供此功能，就必须仿照二维绘画软件中的画笔，实现一个类似的工具。因此，纹理画笔是整个系统中最核心的功能，其实现的好坏会直接影响到整个系统的实用性。

2.2 相关算法研究

此方面的算法最早由Maneesh Agrawala 等人在论文[1]中提出，后来有人把这种方法用于顶点颜色的绘制。而在George ElKoura的论文[2]中，提出了把画笔映射到纹理空间绘制的几种办法：一种是把画笔中心点映射到纹理空间，然后按照通常的二维绘制算法画出画笔。这种方法实现简单，可是对于在世界空间中相邻但在纹理空间中不相邻的三角形，笔迹会出现不连续的现象。另一种是先在屏幕中画出笔迹，再把笔迹映射到纹理坐标。这种办法在表面接近平行于视线的时候，会导致严重的扭曲。除此以外，[3]和[4]采用了另一种办法，即设法把画笔笔刷纹理中的像素直接映射到物体纹理中，此方法效果较好，但计算量比较大。

为了尽可能提高二维绘画的仿真程度，一个趋势是对画笔绘画的物理过程进行建模。例如模拟画笔笔毛的变形过程，模拟颜料的混合，干燥过程。其中[5]提出了3D画笔模型来模拟笔毛变形的过程，[4]则是在此基础上做了各种改进。这些论文提出的方法对今后各种仿真画笔工具的实现会有帮助。

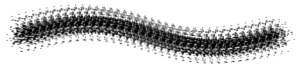
除了用手工绘制的方式来表现物体质感，当前的另一个研究方向是直接模拟物体表面风化的过程。这方面比较好的成果是Yanyun Chen等人的论文[6]。文中提出使用粒子来模拟灰尘、雨水的运动，用点云的形式记录粒子最终停留在物体表面的位置。然后用这些粒子来修改表面的贴图。另外，[7]和[8]模拟了金属生锈的过程，[9]模拟了油漆表面开裂的过程。

在系统的最终实现中，采用的是与[3]、[4]类似的算法，但不同的是，系统中直接利用CPU完成纹理过滤的计算，而不是利用光栅化硬件完成。相比于后者，使用CPU计算的优点是灵活性较大，能够很方便地对效果进行控制，而缺点则是效率偏低，需要对代码进行仔细的优化。

由于时间所限，系统中并没有对画笔绘画或物体表面风化的物理过程进行模拟。但在将来的实现中，若能够把这些算法整合到系统中，则可明显增强系统的实用性。

2.3 纹理画笔的实现

画笔绘制的过程，实质上是把笔刷像素不断映射到物体的纹理像素的过程。在著名的图像处理软件Photoshop中，使用画笔工具绘制时，笔画看起来像是连续的，实际只是不断地把笔刷的图案印到画布上。图 2-1是不同的间距参数下画笔的效果，从中可以明显地看出画笔绘制的实际方法：



a)间距1% b)间距20% c)间距100%

图 2-1 不同间距下画笔的绘制效果

在本系统画笔绘制的功能中，也采用类似的方法。只不过现在画笔的笔刷图案是印到物体表面上而已。根据这种思想，画笔绘制的主要流程如图 2-2：

图 2-2 纹理画笔绘制流程

下面将对其中的每个步骤做详细的介绍。

2.3.1插值

这个步骤的目的是实现调整画笔间距的功能。在最初的设计中，并没有考虑到插值的问题，可是后来发现，鼠标指针移动响应函数捕捉到的鼠标指针位置之间的间隔比想象中的大得多。于是，采用图2-3中所示的方法对鼠标指针位置进行采样。

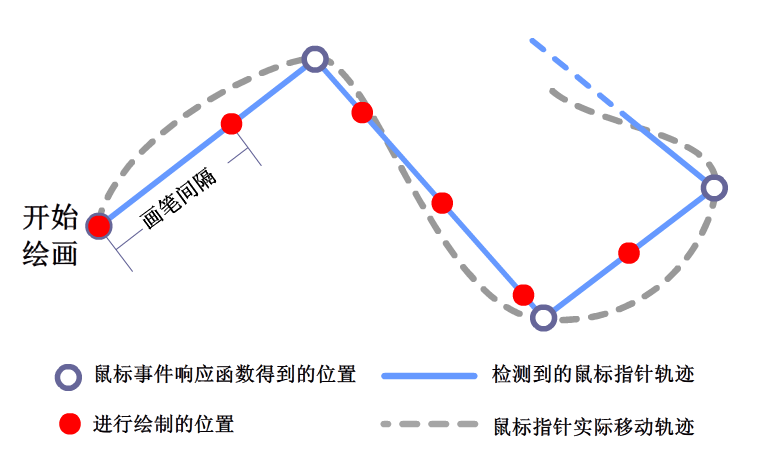


图2-3 插值

图2-3中的算法实质上是在鼠标拖动时，求出鼠标指针经过的路径的近似长度（图中实线的长度），然后根据设定的画笔间隔，确定要进行绘制的位置。

上述方法可以很好地对付二维绘画的情况，可是，如果是在三维物体上绘画，情况还要复杂一些。如图2-4所示，在三维物体上绘画时，即使屏幕上的要绘制的点是间隔均匀的，可是映射到物体表面之后，就不一定了。因此应该改为以鼠标映射到物体上的轨迹（就是图中物体上连接各点的粗实线）来确定画笔的绘制点。

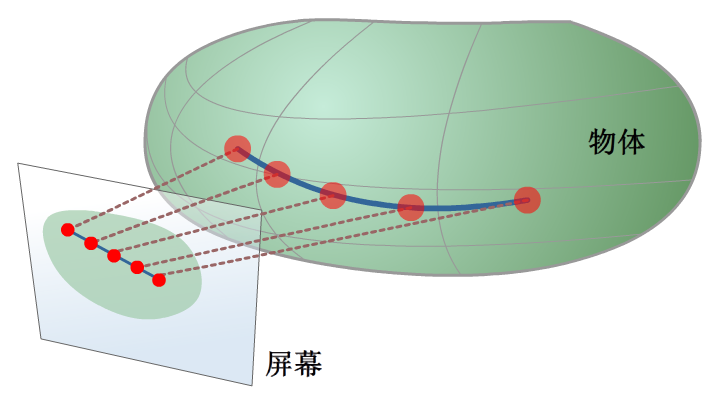


图2-4 物体表面凹凸引起的不均匀映射

为了把鼠标指针映射到物体表面，最直接的办法，是通过读取Zbuffer，获得鼠标指针所在点的深度，根据观察矩阵换算出点的实际位置。在系统的实际实现中，由于需要更灵活的控制，没有采用直接读取Zbuffer的办法，而是利用自定义的着色器程序，把深度信息保存在渲染结果的一个通道中。

2.3.2 画笔定位

确定了要绘画的位置之后，下一步就是把笔刷放到物体表面上。如图2-5所示，首先建立一个坐标系。其中，x方向为画笔轨迹的切线方向，z方向为物体表面法线，y方向则为z与x的叉乘方向。为了获得物体表面的法线方向，需要像获得深度值一样，预先把从用户角度观察到的物体表面法线渲染到一张纹理中，然后通过读取这张纹理得到物体表面的法线数据。

计算好局部坐标系之后，下一步就是根据建立的坐标系放置一个正交摄像机，把物体的表面信息渲染到另一张纹理中。这些信息包括物体的纹理坐标，表面的三角形序号，物体表面法线，还有表面的深度。在下一步的绘制过程中，会利用到这些信息。

值得注意的是，为了实现更加丰富的效果，可根据一组用户可以调节的参数计算出摄像机的位置。而不只是把摄像机固定在坐标原点。例如，摄像机可根据用户设定的范围选取一个随机位置，以模拟出颜料散布的效果。

在系统的实现中，控制摄像机放置的参数被集中到画笔面板中，用户可以通过调节参数来实现丰富的笔刷效果。这些参数的作用会在第四章详细介绍。

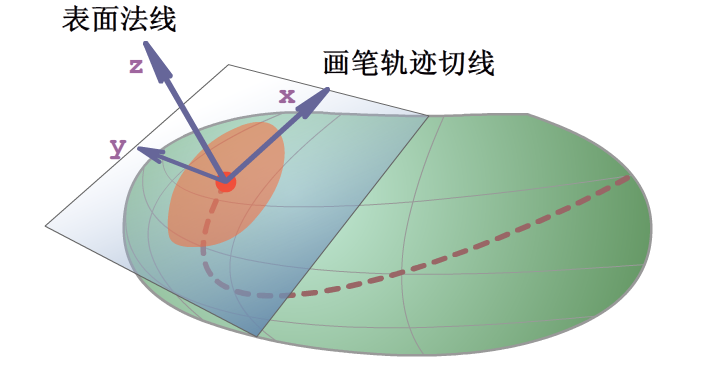


图2-5 笔刷坐标系

2.3.3像素映射

这是整个绘画过程最关键的一步，也是最难实现的一步。在这一步中，需要把用户设定的画笔颜色、反射等属性画到物体表面上。其基本流程如图2-6。为了做出丰富的笔触效果，需要一张笔刷纹理来记录笔刷的形状、颜色等属性。而在上一步，已经根据局部坐标放置好摄像机，同时把物体的纹理坐标等表面信息渲染到一张纹理中，这张纹理称为表面信息纹理。于是，可以遍历一次表面信息纹理，对其中的每个像素，先在笔刷纹理中找到对应位置的颜色、反射率等属性，然后根据像素中存储的纹理坐标值，找到并修改对应的纹理像素。当所有像素都修改完了之后，把修改过后的纹理更新到openGL，就可以即时看到修改过后的效果。

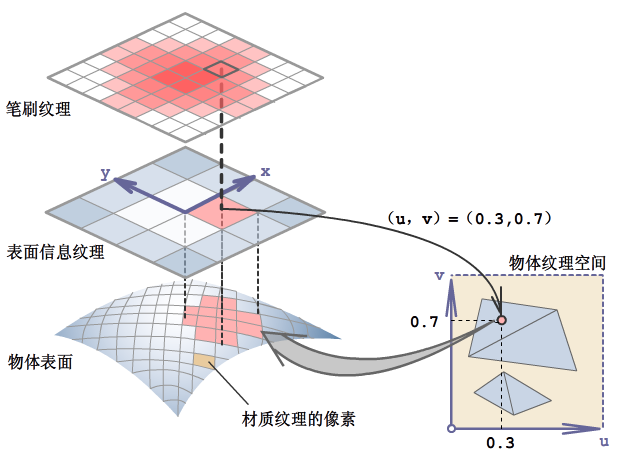


图2-6 像素映射

在实际的实现中，还有一个问题需要处理。就是表面信息纹理的一个像素，往往并不准确地对应物体纹理的一个像素。在图2-6中，实际上是对应了8个像素。这时候，如果只根据物体的纹理坐标修改其中的一个像素，必然会导致很多应该修改的像素没被修改。在用户看来，笔画中有很多空洞没被填充。这会严重影响到绘制的最终效果。因此，应该事先换算出表面信息纹理的一个像素实际对应物体材质纹理的像素范围。以下对这个过程进行说明。

为了简化计算，首先假定，表面信息纹理的每个像素只覆盖一个三角形。在绝大多数情况下，物体表面的三角形大小都比这些像素要大，甚至比整个笔刷要大，所以这个近似都是合理的。

由于有了这个近似，整个过程大概分成两部分：首先把笔刷的局部坐标变换到三角形的纹理空间中（如图2-7），然后根据所占的像素进行填充。

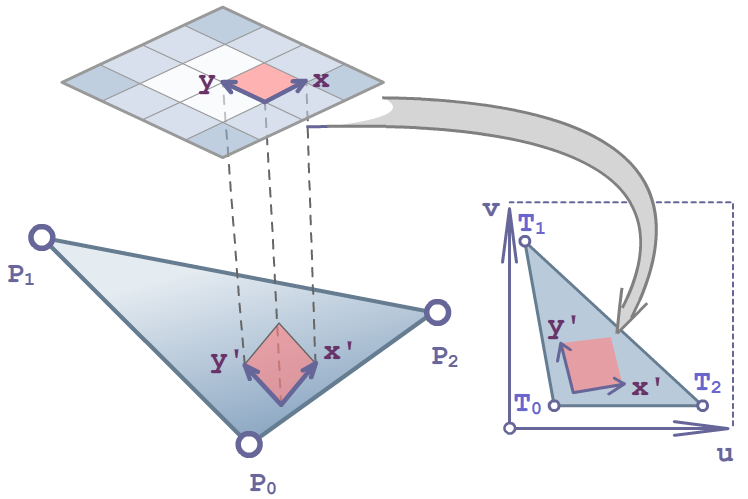


图2-7 像素坐标变换

如图2-7， 图中 **、、**为三角形三个顶点的局部坐标，**、、**为其对应的纹理坐标。变换的过程实际上是反过来的，先把纹理坐标系的基向量**u**、 **v**变换到笔刷局部坐标系，然后求解相反的变换。由于在画笔局部坐标上放置的摄像机使用正交投影，因此，所有变换均为线性的。

假定投影矩阵为，观察矩阵为，物体变换矩阵为。**Pi**为**P0~P2**中的一点，Pi’为变换到画笔坐标系之后的一点，显然有：

**(i=0,1,2)** (1)

实际上，我们只关心三角形顶点之间连线的方向，因此，上式可以换成顶点之间的差值。假设 ，**， , ,**则有：

**(i=1,2)** (2)

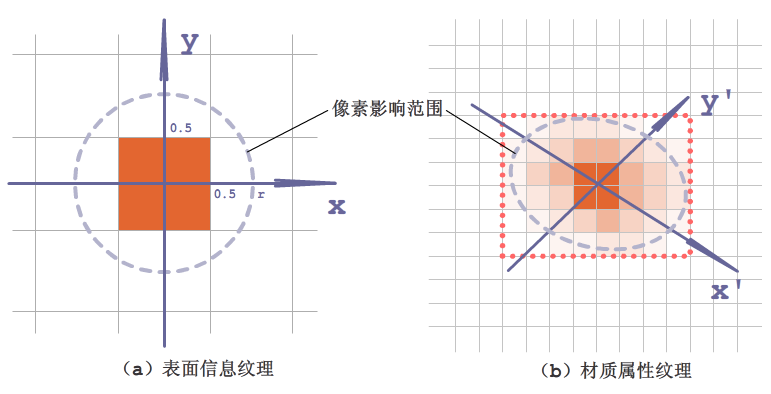
其中，下标u、v分别表示纹理坐标差值的u、v分量。由此式就可以求出纹理坐标系基向量映射到笔刷局部坐标系之后的坐标。

下一步，用类似的方法把笔刷局部坐标系映射到纹理坐标系，利用下面公式：

(3)

即可求出笔刷局部坐标系的基向量**x**, **y**映射到纹理坐标系的结果。其中x的坐标为 ,y的坐标为,其中nx，ny分别为表面信息纹理在各自方向的像素个数。

接下来需要根据映射后的向量填充相关的像素。如图2-8a，计算好变换后的坐标后，还要确定其在材质属性纹理中影响的像素范围。方法是预先在表面信息纹理中定义一个圆，这个圆映射到材质属性纹理空间后是一个椭圆，为了简便起见，把在椭圆包围框（图2-8b中的虚线框）内部的像素作为要影响的像素。然后，对于这些像素，确定它们在表面信息纹理坐标系中的坐标，再以其到坐标原点的距离为参数，代入一个衰减函数（如高斯函数）得到权重，根据权重修改颜色。



a)表面信息纹理 b)材质属性纹理

图2-8 像素过滤

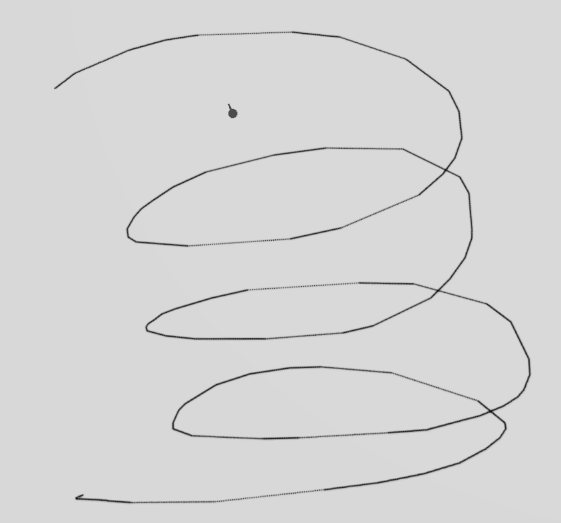
2.3.4像素更新

在最简单的情形下，修改完纹理像素之后，把修改后的整幅纹理传到openGL，即可看到更新后的结果。然而，这样做效率很低，因为在大多数情况，被修改的像素只占了整张纹理的一小部分，很多像素是没有变化的。如果全部都进行更新，就会浪费很多的计算时间，尤其是当物体具有多个图层的时候，每次更新都需要重新对图层进行混合，计算量十分可观。而且在绘画的过程中，如果不断地把整幅图像传输到显卡，也会大大地消耗带宽。这两个因素都会大大降低纹理更新的速度，使得交互很不流畅，从而明显降低系统的可用性。

针对这种情况，解决的办法是用一个数组记录要更新的像素，在更新的时候只处理被影响到的像素，同时计算包围这些像素的最小矩形，在更新到openGL的时候，只传输矩形内部的像素。实际测试表明，采取这样的措施之后，更新的速度明显改善。

2.4 结果与分析

本节通过对比不同画笔大小下的绘制效果，从而指出当前算法的缺陷。



1. 大小为0.1 b) 大小为1

图2-9 不同笔刷大小下的绘制轨迹

图2-9展示了不同笔刷大小下画笔绘制的轨迹。可见当笔刷较小时，笔迹比较平滑；而当笔刷较大时，笔迹则比较生硬，同时，交互的速度也明显变慢。

根据上述结果，联系上文提及的插值算法，不难看出，图2-9b中生硬的笔迹是由于鼠标事件响应函数捕捉到的点过于稀疏而产生的。而导致这种情况，只可能是因为绘制时间过长。交互速度的变慢也证明了这一点。这说明当前的绘制算法效率还不够高。

通过对当前算法的分析，发现后两步，即像素映射和像素更新消耗的时间较多。对于像素映射，其效率主要受表面信息纹理的分辨率影响。而对于像素更新，最大的影响因素自然是要更新的像素数量。在当前的实现中，用户可以调整表面信息纹理的分辨率，从而在速度和质量之间得到某种平衡。但对于像素更新阶段的效率问题，暂时还没有有效的解决办法。

在今后的改进工作中，若要进一步提高绘制效率，最根本的解决办法是对算法进行优化。例如，可以把一些计算结果预先存起来，避免重复计算。同时，可以把鼠标指针位置的捕捉跟像素绘制分成两个线程处理，同时根据负载情况调整绘制的质量；还可以采用更高阶的插值方法，以得到更加平滑的轨迹。

2.5 本章小结

本章对纹理画笔算法的具体实现及其实际效果进行了比较详细的介绍。纹理画笔的绘制，实质上是把笔刷像素映射到物体纹理空间的过程。在这个过程中，除了涉及坐标变换的问题，为了产生比较好的效果，还需要对纹理进行过滤。而如何把整个流程有效率地实现，则是系统设计中的难点。总的来说，在当前的实现中，较好地解决了上述问题。

第三章 纹理接缝消除算法的设计与实现

3.1 纹理接缝的产生原因

在系统最初的实现中，当对物体表面高度进行编辑的时候，在纹理坐标的接缝处会产生明显的裂缝。如图3-1所示，在球的右下角箭头所示区域有明显的裂缝。

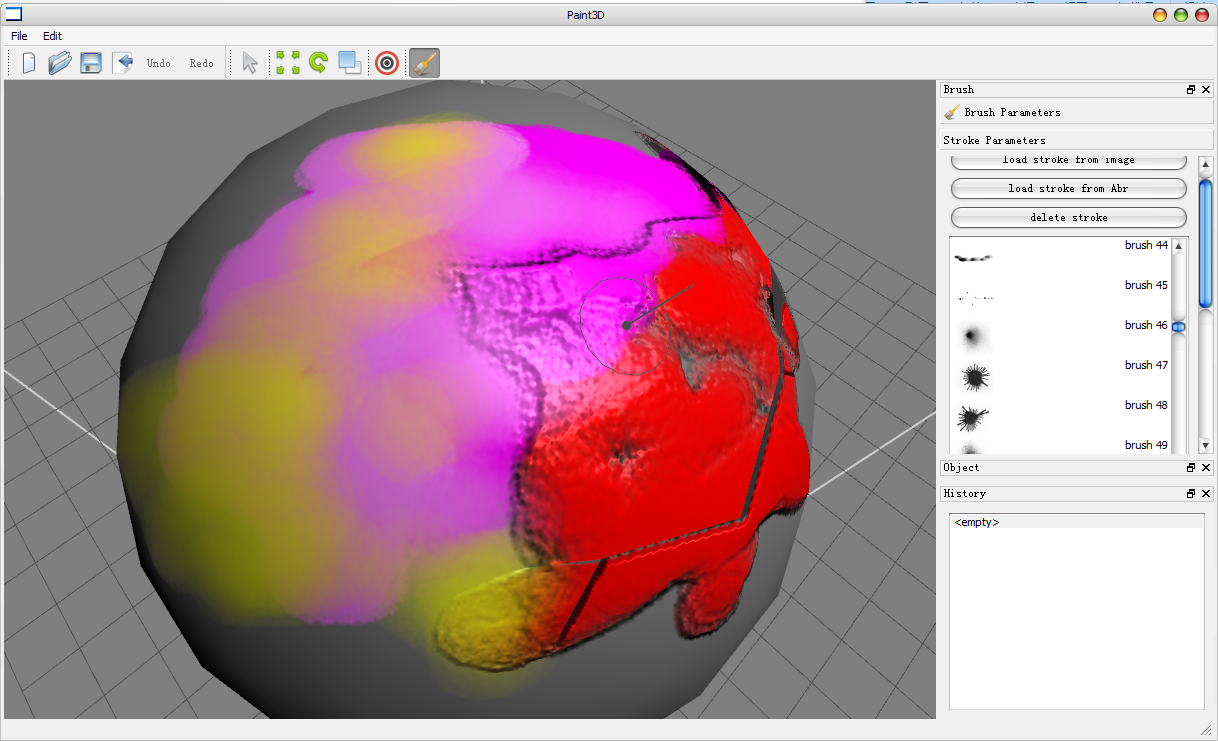


图3-1纹理裂缝

导致裂缝出现的原因如图3-2。为了真实地表现物体表面的厚度变化，需要预先存储一张高度图，在对物体进行渲染的时候，对屏幕每个像素进行类似光线追踪的操作（称为Relief Mapping），利用数值方法（如二分法）求出像素对应的射线vt跟考虑高度之后的物体表面的近似交点（图3-2中带x标记的点）。但是，当物体的uv空间分成几大块，而射线又刚好到达块边界上的一点（如图中，射线到达了3号点）的时候，只有穿过边界，找到这一点在相邻的块对应的位置，才能继续追踪，找到射线与物体表面的实际交点。但是，在不对高度图进行特殊处理的情况下，光线是没有办法越过边界找到正确的交点的。

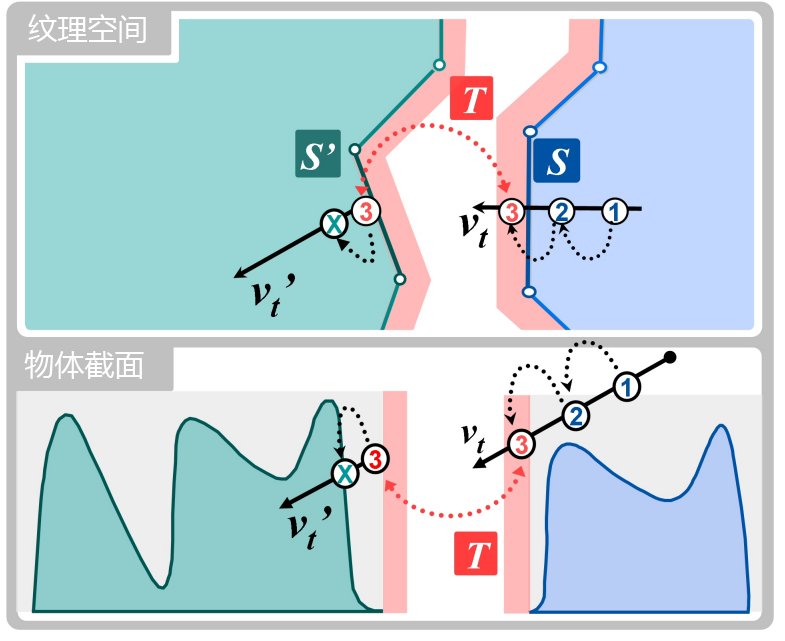


图3-2 越过边界的光线

3.2 现有算法研究

实际上，即使是传统的材质编辑方法，由于各种各样的原因，纹理接缝一直是需要解决的问题。针对此问题，最新的成果在论文[10]中。该文中假设模型均由四边形面片组成，而且，每个四边面分别对应纹理贴图中的一个矩形像素区域。相对于传统的方法，这种办法的特点在于贴图展开的基本单位不是三角形而是四边形。有了这两个约束，面与面之间贴图接缝问题变得容易解决。可是对于现存的大量三角形模型，至少首先需要把三角形网格全部转换成四边形网格，而这一步骤一来实现比较麻烦，二来对于创作人员也会造成某种不便。因为某些细节，例如褶皱衣服上的线，在展开后的纹理空间绘制，反而会比直接在物体表面绘制更加方便。美术人员若在展开uv的时候仔细考虑物体的结构，合理分配接缝，则可以大大简化随后的绘制工作。而Ptex要求每个面都是一个孤立的纹理，如果相邻的面的纹理分辨率不一样，那就没办法实现这样的编辑方式。基于上述考虑，在系统中没有采用Ptex的技术。在这篇论文之前，另一篇相关的论文[11]，提出先把三角形拼成一个个四边形的片，然后每片采用Ptex类似的技术。总的来说，这两篇论文的方法都不太适合整合到系统中，但其采用的纹理过滤方法可能会有参考价值。

除了前述一些论文提供的方案之外，[12]提出了一种办法，就是在边界外侧记录一些信息，使得光线在跟踪的过程中能查找出到相邻块的边界的变换矩阵，从而转到相邻对应的块中。这种办法可以很好地解决接缝的问题。

然而，由于本系统的目的是让用户编辑并输出材质贴图到三维软件的材质编辑器中，而目前主流的三维软件在渲染的过程中，仍然采用传统的纹理查找和过滤方法（例如，对相邻的像素做双线性插值），并没有考虑纹理被分成多块时的情形。因此，即使应用了这种办法能够在系统中能够显著减少接缝的出现，可是由于边界信息无法被其他软件识别，当输出的贴图被导入三维软件渲染的时候，接缝处仍会出现瑕疵。

考虑到上述因素，在系统实际的实现中，并没有采用上述比较有效的接缝消除办法，而是采用了相对传统的做法，就是在边界之外的一定范围内，把边界内侧附近的像素的颜色填充进去。这个办法无法完全消除接缝，可是，由于不需要三维软件的特殊支持，在当前的技术条件下，是比较可行的。

3.3 接缝填充算法具体实现

下面介绍具体的实现方法。

如图3-3，图中A和B是物体贴图坐标空间中两个孤立的块。和，和各自对应局部坐标的同一个顶点。因此，边和边实际上是相邻的。为了确定两条边界的外侧各自需要填充的区域**P1**、**P2**，首先仿照网格的顶点法线的做法，对于边界上的一个顶点，根据与其相邻的两条边的法线构造该顶点的法线（即图中的）。有了这些法线，就可以构造边界外侧被填充像素的区域**P1**、**P2**，同时，只要把法线反转，用同样的办法也可以构造边界内侧的像素取样区域 。于是，问题就变成把**P1**的像素对应到，以及把**P2**的像素对应到。假设这种对应关系成功建立，在绘制的时候，当遇到靠近边缘的像素，如中的一个像素，只需要根据预先建立的查找表，找到其在**P2**中对应的像素并填充即可。也就是说，在绘制的过程中，并不需要大量额外的计算，这一点保证了交互能够流畅进行。

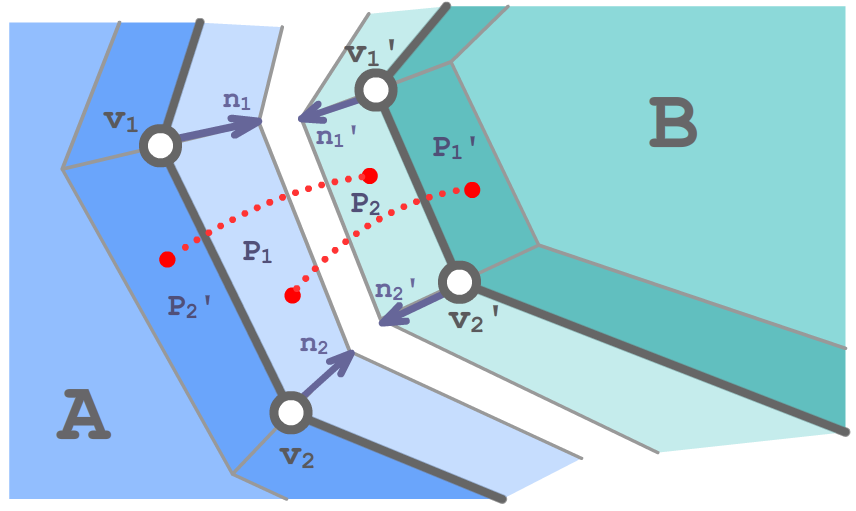


图3-3 区域像素的对应

如何把像素映射到对应的区域呢？从图中可以发现，所有需要进行映射的区域均为四边形，所以，我们可以先对四边形进行参数化，求出映射前的点**P**在四边形中的参数坐标，然后把这个坐标代入目标四边形，即可求出映射后的点。

关于四边形的参数化，已经有很多研究[13][14][15]，本系统采用最简单的线性插值方法。

如图3-4a，**P**为四边形内部的一点，需要求出**P**在四边形参数空间中的坐标。图中，定义四边形三条边的方向向量分别为**U**、**V1**、**V2**，于是，可以用下式定义四边形中一点**P**的参数坐标：

(4)

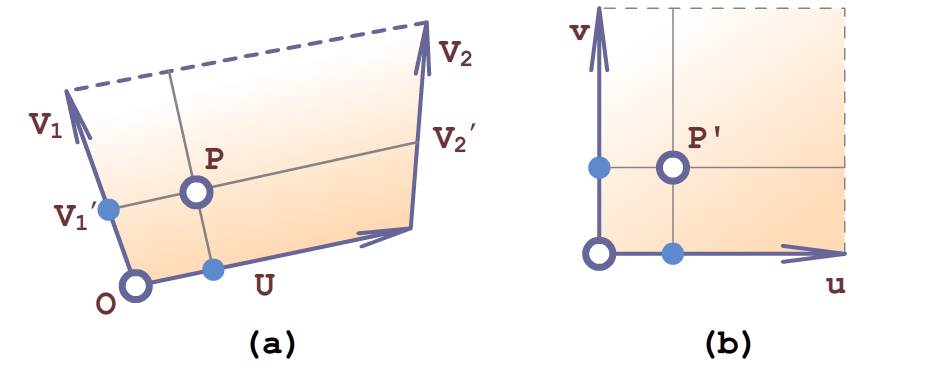
(5)

(6)

根据上述三式，有

(7)

这实质上是一个二元二次方程组，通过求解这个方程组，即可求出P在参数空间的坐标x和y。



1. 四边形坐标空间 b) 参数空间

图 3-4 四边形插值

在实际的实现中，还需要考虑退化的情况。例如，当四边形是个梯形时，方程组的次数为1。这时候，就不能直接应用二次方程的求根公式，否则会出现除数为0的错误。

求出了**P**在参数空间的坐标之后，把**P**、**O**、**U**、**V1**、**V2**这些点换成另一个四边形的对应点，代入上述公式，就可以很方便地求出映射后的点了。

3.4 结果和分析

图3-5显示了同一个模型在采用接缝填充算法前后的绘制效果。图中画笔的厚度参数为1。可以看到，当前的填充算法可以明显地减小接缝带来的瑕疵。

表3-1 采用填充算法前后的效果对比

|  |  |
| --- | --- |
| **没有采用填充算法** | **采用填充算法** |
|  |  |

表3-1 采用填充算法前后的效果对比（续）

|  |  |
| --- | --- |
| **没有采用填充算法** | **采用填充算法** |
|  |  |
|  |  |

但是，在实际的运行中，发现此算法需要比较多的预处理时间。这是由于读入的模型文件往往只是给出了坐标轴和面的坐标索引，因此，需要首先对数据进行处理，找出纹理坐标空间中各个部分的边界，以及位于不同边界上的公共点。然后，还要计算边的法线，并进一步计算公共点的法线，然后构造四边形。这些准备步骤看起来很直观，可是实现出来是需要花点功夫的；而最核心的像素映射算法，实现起来则相对简单。当前的实现中，由于准备数据的部分比较复杂，所以消耗了相当一部分预处理时间，用户导入模型之后，往往需要等上10多秒才能继续操作，这一点是需要在后续的工作中完善的。

经过纹理裂缝填充算法处理之后，物体在接缝处，尤其是在一些顶点附近仍然会有一些瑕疵，如图3-5中黄色圈里面的一些毛刺。可是大条的裂缝已经没有了。在之后的进一步实现中，还需要对算法进行进一步的改进，以消除这些瑕疵。

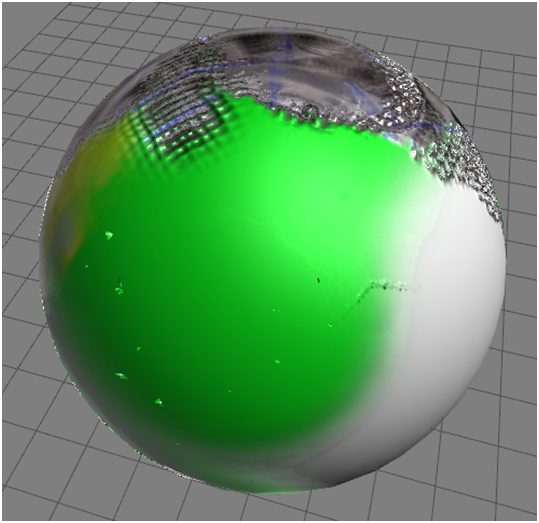


图3-5 填充接缝后模型表面的瑕疵

3.5 本章小结

本章对纹理接缝的产生原因，以及当前系统采用的接缝填充算法进行了比较详细的介绍。使用这个算法，能够明显减小接缝带来的瑕疵，但还不能完全消除接缝。在今后的工作中，还要寻找更有效的接缝消除算法。

第四章 系统实现与测试

4.1 系统总体架构设计

系统架构如图4-1，本系统分为三层，由顶层开始，依次为界面操作层，场景管理层和底层操作层。

模型数据管理

画笔操作、画布管理

历史记录

openGL显示与读取操作

摄像机相关操作

物体增删、导入、保存

鼠标事件响应

自定义控件

控件消息处理

图4-1 系统架构

4.1.1 界面操作层

界面操作层主要完成两方面的任务：一个是用户界面的显示，另一个是用户操作的处理。对于用户界面的显示，由于Qt提供了比较方便的界面编辑方法，很多代码实际上由Qt Designer自动生成。而一些自定义控件，例如图4-2中带滑块的微调器，以及右侧的曲线图，则需要自己实现。



图4-2 自定义控件

而用户操作的管理主要包括两方面的内容，一个是界面控件产生的消息，另一个是用户的鼠标键盘输入。前者控制逻辑相对简单，利用Qt的signal和slot可以很好地处理，而后者逻辑则相对复杂。因为在三维软件中，用户的很多操作，例如摄像机的旋转，物体的移动，都是通过鼠标和键盘配合来完成的。这样在处理同一个消息，例如鼠标左键按下的消息的时候，就需要根据当初场景所处的不同状态来确定相应的操作。例如，当用户处于选择物体模式，按下鼠标左键应该把物体选中。而当此时用户按住了Alt键，则应该旋转摄像机，如果用户处于选择面模式，按住鼠标左键拖动后，屏幕中应该显示一个方框来表示当前的选区。在系统的最初实现中，所有与用户的鼠标，键盘操作交互有关的状态保存在Scene类中，但是，Scene类同时还需要维护场景数据，而且这两种任务之间的联系并不是十分密切，于是在最后的实现中，把这些操作和状态移到了GLViewWidget类之中。

4.1.2场景管理层

这一层只由一个Scene类构成。在Scene类中维护了场景的各种数据，例如物体的几何数据，与三维绘画相关的画布、笔刷数据，openGL的显示状态数据以及用户操作的历史记录数据等。Scene类的行为是被动地接受上一层的操纵，然后把这些操作传递给下一层。其本身只保存当前被用户改变后的场景的状态，而不保存用户当前的输入状态。例如，当用户选中一个物体之后，这个物体应该显示出线框，如图4-3，可是对于Scene类来说，它只知道当前场景中有一个物体需要以线框显示，而不知道这是因为用户选择了这个物体。后者是保存在界面层的GLViewWidget中，如前所述，这个类把用户鼠标、键盘的操作转化成对Scene类的操作。

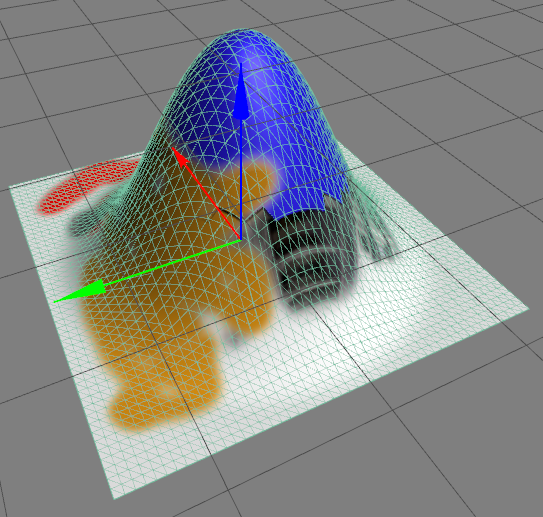


图4-3 选中的物体显示线框

4.1.3底层操作层

这一层由各种场景数据类组成。其中包括，维护物体网格和变换数据的Mesh类，维护物体表面数据的Canvas类，控制笔刷操作的Painter、Brush类，实现反射效果的CubeMap类，和保存场景历史记录的类等等。这些类被动地接受Scene类的操控，自身对当前场景所处的状态一无所知。

4.1.4 主要的类及其功能

以下列出了系统一些比较重要的类，它们的功能以及各自所在的层。

表3-1 一些重要的类及其功能

|  |  |  |
| --- | --- | --- |
| 层 | 类名 | 功能 |
| 界面操作 | BlendCurveGraphWidget | 实现图4-2中右侧的混合曲线的显示 |
| 界面操作 | ColorSelectButton | 实现颜色选择。 |
| 界面操作 | SliderSpinnerGroup | 实现图4-2中央带滑块的微调器。 |
| 界面操作 | GLViewWidget | 处理屏幕的显示，以及用户的鼠标键盘操作 |
| 界面操作 | Manipulator | 抽象类，包含TranslateManipulator、RotateManipulator、ScaleManipulator三个子类，负责把用户的鼠标操作转换成对物体的平移、旋转和缩放操作。 |

表3-1 一些重要的类及其功能（续）

|  |  |  |
| --- | --- | --- |
| 层 | 类名 | 功能 |
| 场景管理 | Scene | 管理场景数据，包括物体的增加删除，重命名，以及场景的打开保存等操作。 |
| 底层操作 | Camera | 管理摄像机的平移、旋转、缩放等操作。负责计算当前的观察矩阵和投影矩阵。 |
| 底层操作 | CubeMap | 负责立方体贴图的读入和转换，以及相关openGL数据的维护 |
| 底层操作 | GeometryExposer | 这是整个软件比较关键的一个类。它的功能是把物体的表面信息渲染到openGL的帧缓存中，然后通过从显存中读取帧缓存，把物体的表面信息提取出来。 |
| 底层操作 | Mesh | 负责物体几何数据的管理，以及openGL Vertex Buffer Object的维护。每个Mesh会有一个Canvas对象，存放物体被绘制的表面像素。 |
| 底层操作 | ObjReader | 负责从obj格式文件读取顶点数组，放入Mesh对象 |
| 底层操作 | RenderableObject | 抽象类，表示场景中用户可见的物体。通过这个类实现一些物体统一的操作，例如渲染物体的几何信息，渲染物体的外观等。 |
| 底层操作 | ObjectTransform | 这个类负责管理物体的变换（移动、旋转、平移）参数。同时提供一些操作的接口，方便各个操纵器（Manipulator）对物体进行控制。 |
| 底层操作 | Brush | 画笔笔刷，这个类负责把颜料绘制到物体表面上。前面所述的绘制算法，其核心部分就是在Brush类里面实现。 |
| 底层操作 | Painter | 负责根据鼠标位置，放置笔刷。 |
| 底层操作 | Canvas | 表示物体表面的纹理。负责管理各个图层的混合，以及纹理数据的更新。 |
| 底层操作 | CanvasSeamFiller | 这个类实现前述的纹理接缝消除算法。根据物体的纹理坐标生成一张像素查找表，绘画的时候利用这张表来消除接缝。 |

表3-1 一些重要的类及其功能（续）

|  |  |  |
| --- | --- | --- |
| 层 | 类名 | 功能 |
| 底层操作 | PaintPicker | 抽象类，负责绘制时材质颜料的拾取工作。有三个子类继承它，分别是PurePaintPicker，PlanePicker和ImagePicker。PurePaintPicker可以拾取纯颜色值，ImagePicker可以拾取图片中的颜色，而PlanePicker则是一个在场景中可见的平面，用户可以把平铺在上面的图片投射到物体表面。 |
| 底层操作 | Stroke | 画笔笔触类，可以通过加载不同的图片模拟不同的绘制效果。 |

4.2 系统环境选择

本系统采用Qt界面库，因此，在这里先介绍Qt的一些特性。

4.2.1 Qt概述

Qt 是一个跨平台的 C++ 图形用户界面库，由挪威 TrollTech 公司于1995年底出品。Trolltech 公司在 1994 年成立，但是在 1992 年，成立 Trolltech 公司的那批程序员 就已经开始设计 Qt 了，Qt 的第一个商业版本于 1995 年推出。

2008年1月31日，Nokia公司宣布通过公开竞购的方式收购TrollTech公司，旗下包括Qt在内的技术都归入Nokia旗下。并且Nokia针对自己的移动设备平台规划的需要，将Qt按不同的版本发行：

Qt[商业版](http://baike.baidu.com/view/805792.htm" \t "_blank)：提供给商业软件开发。它们提供传统商业软件发行版并且提供在协议有效期内的免费升级和技术支持服务。

Qt开源版：仅仅为了开发自由和开放源码软件， 提供了和商业版本同样的功能。GNU通用公共许可证下，它是免费的。

基本上，Qt 同 X Window 上的 Motif，Openwin，GTK 等图形界 面库和 Windows 平台上的 MFC，OWL，VCL，ATL 是同类型的东西，但是 Qt 具有下列优点:

1. 优良的跨平台特性。Qt支持下列操作系统: Microsoft Windows 95/98， Microsoft Windows NT， Linux， Solaris， SunOS， HP-UX， Digital UNIX (OSF/1， Tru64)， Irix， FreeBSD， BSD/OS， SCO， AIX， OS390，QNX 等等。
2. 面向对象。Qt 的良好封装机制使得 Qt 的模块化程度非常高，可重用性较好，对于用户开发来说是非常 方便的。 Qt 提供了一种称为 signals/slots 的安全类型来替代 callback，这使得各个元件 之间的协同工作变得十分简单。
3. 丰富的 API。Qt 包括多达 250 个以上的 C++ 类，还提供基于模板的 collections， serialization， file， I/O device， directory management， date/time 类。甚至还包括正则表达式的处理功能。
4. 支持 2D/3D 图形渲染，支持 OpenGL
5. 大量的开发文档
6. XML 支持

在以上的优点之中，Qt的一个与众不同的特征是，它提供signal和slot概念，这是一种安全可靠的方法，它允许回调，并支持对象之间在彼此不知道对方信息的情况下，进行合作，这使Qt非常合适于真正的组件编程。因此，Qt具有模块设计和组件或元素的可重用性的特点。一个组件不需要知道它的内容和用途，通过signal和slot与外界通信、交流。而且，所有Qt的组件都可通过继承来扩展功能。

在当前的大部分界面库中，大多数采用回调函数来实现程序的消息响应。所谓的回调是指：由程序员自己定义一个函数，并告诉系统何时为何调用。系统在需要的时候，就会调用你定义的函数，并提供处理参数。虽然回调函数能够实现用户之间的交互。但是，回调函数非常复杂，容易混淆，又难以理解(至少大部分编写Qt的工作人员或者程序员有过这样的想法)。因此，Qt的开发者使用另一种方法来完成这一工作。这种方法依赖于Qt特有的两个功能，signal和slot。使用这种新方法是非常简单的，只需要编写一行代码就能够将用户事件和程序事件连接起来。Signal和slot的工作方式如图4-4所示：

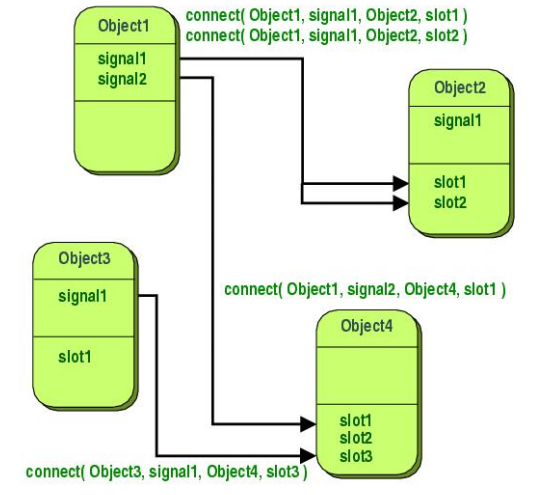


图4-4 Qt的信号和槽

另外，由于Qt是一种跨平台的GUI工具包，所以，它对编程者隐藏了在处理不同窗口系统时的潜在问题。为了将基于Qt程序更加方便，Qt包含了一系列类，该类能够使程序员避免了在文件处理、时间处理等方面存在依赖操作系统方面的细节问题。

4.2.2 Qt在系统中的运用

Qt不仅提供了极其方便的界面开发库，针对各种应用，Qt还提供了对应的一些工具类。由于本系统是一个绘图程序，因此主要使用了Qt提供的图形库。

在二维绘图方面，Qt提供了一整套工具，例如QPainter类，在任意一个绘图设备上都可以润色图形。绘图设备包括组件、像素映射、图形文件和打印机，相同的代码可以用在4种不同类型的设备上。 QPainter类支持复杂的同等系统的转换，很容易在所有平台上画旋转文本和像素映射。在本系统的实现中，需要使用一些二维绘画的功能，而QPainter类恰恰提供了这些功能，这极大地减轻了编程负担。而QImage类则对各种图片格式的读写做了很好的封装，使得程序员可以不必考虑各种图片格式的具体细节，而是专注于图片处理工作本身。在本系统的实际实现中，经常涉及一些对图片文件的读写操作， 而QImage类极其方便的文件读写功能使得这些工作变得轻而易举。

在三维绘图方面，Qt提供了QGLWidget类,使用该类能够绘制2D/3D图形。用QGLWidget就像用一个Qt组件一样方便。这比纯粹的用OpenGL做的3D图形更好。同时针对openGL的一些特性，例如shader语言、Framebuffer Oject等，Qt都提供了相应的封装。这使得使用这些功能变得异常方便。对于三维绘图过程中涉及到的向量、矩阵运算，Qt也提供了相应的封装，而且这些封装跟上述的openGL功能类能够无缝地连接，这大大简化了开发人员的工作。

另外，Qt还实现了一些与C++标准库类似的容器和算法，如类似std::vector的容器QVector，类似std::map的QMap，这些容易既提供了跟标准库一样的接口，也提供了Java风格的迭代器访问方式，同时还有一些Qt的特色功能。在系统中，大量运用了QVector、QHash、QSet这些Qt容器。

4.3功能模块描述

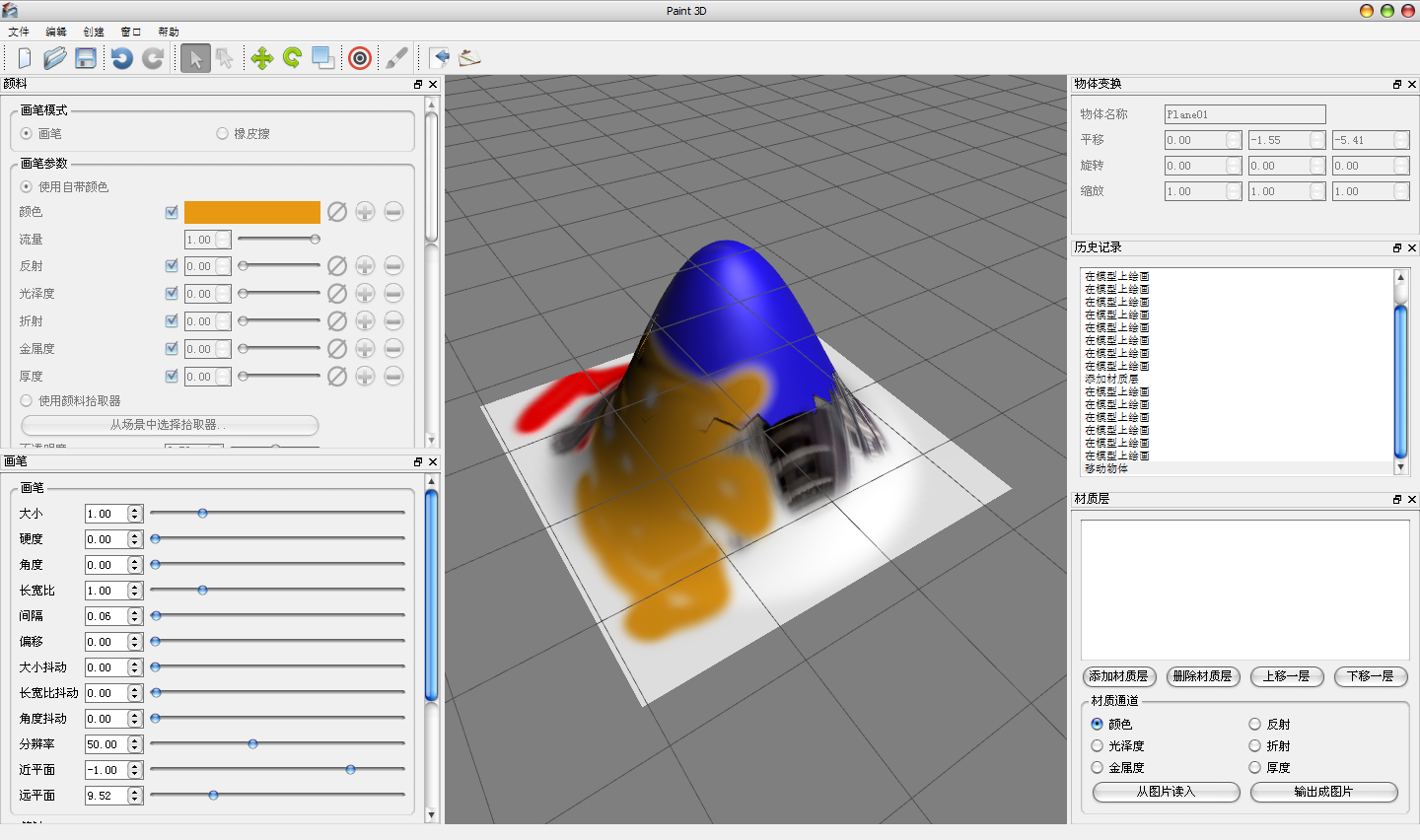


图4-5 程序的用户界面

图4-5是程序的用户界面。从图中可以看到，除了通常的菜单栏和工具栏之外，程序中央是显示三维物体的视图。围绕着中央视图是一些参数面板。分别是颜料面板、画笔面板、物体变换面板、历史记录面板和材质层面板。下面会对软件的基本操作和这些面板各自的功能进行详细说明。

图4-6为系统的主要功能模块，下面对各个模块进行详细的描述。

图4-6 主要功能模块

4.3.1 文件操作

本系统允许用户把编辑后的场景保存为一个p3d文件。因此，文件操作模块提供了新建、打开、保存文件这些常用的功能。其使用方法跟大多数软件一致。

另外，系统允许用户从外部导入.obj格式的三维模型。

4.3.2 视图操作

视图操作主要包括摄像机的旋转，缩放和平移、物体的选择、物体面的选择、平移、旋转和缩放物体，以及画笔的绘制。在系统中，这些操作主要通过鼠标拖动和键盘配合完成。以下具体介绍这些功能。

摄像机旋转：按住Alt键和鼠标左键拖动，就可以旋转摄像机。

摄像机平移：按住Alt键和鼠标中键拖动，就可以平移摄像机。

摄像机缩放：按住Alt键和鼠标右键拖动，或者滚动鼠标滚轮，就可以缩放摄像机。

摄像机对焦：选中场景中的一个物体，然后按下F，就可以把摄像机移到物体前面，从用户的角度来说，就是可以对准物体观察。

选择物体：点击物体选择工具，然后在要选中的物体上点击，就可以选中物体。物体被选中之后，表面的边会被显示出来。

选择物体表面：点击面选择按钮或者使用快捷键A，进入面选择模式，进入之后，视图背景会变成紫色，表示当前模式为面选择模式。这时候按住鼠标拖动，屏幕上会显示一个矩形，鼠标松开后，跟这个矩形相交的所有面会被选中。

在面选择模式下，用户也可以通过单击鼠标选择面。如果想在现有的基础上再选中面，可以按住Ctrl键点击或框选新面，如果想去除部分当前选中的面，可以按住Shift键点击或框选。

平移物体：点击平移物体按钮，或者按快捷键W，可以进入平移模式。在平移模式下，选中的物体上会显示一个坐标系，当鼠标移动到坐标系的一个轴向上面时，对应的轴向会变成黄色，这时候，拖动鼠标，就可以移动物体了。

旋转物体：点击旋转物体按钮，或者按下快捷键E，可以进入旋转模式。在旋转模式下，选中的物体会显示一个球，使用跟移动工具类似的方法，在某个圆环上拖动鼠标，就可以旋转物体了。

缩放物体：点击缩放物体按钮，或者按下快捷键R，可以进入缩放模式。在缩放模式下，选中的物体会显示一个类似平移时的坐标系，使用跟移动工具类似的方法，在某个轴向上拖动鼠标，就可以缩放物体了。

画笔绘制：这是整个程序最核心的功能。在当前的实现中，导入模型之后，就可以直接在模型上绘画了。要在物体上绘画，首先需要选中物体，点击画笔工具或者按下键盘的B键，进入绘画模式。进入绘画模式之后，视图的背景会变成深蓝色，如图4-7a。然后按住鼠标拖动，就可以在模型上绘制了，如图4-7b。



a) 进入绘制模式 b)在模型上绘制

图4-7 绘制物体

4.3.3 颜料控制

如果要做出令人信服的效果，还需要在各种属性面板中细心调节各种参数。其中，颜料面板负责控制画笔绘制的效果。以下逐个介绍这个面板的参数。

图4-8显示了整个颜料面板。下面将对各种参数做讲解。

在面板的最上方，可以选择两种模式：画笔和橡皮擦。跟普通的二维绘画软件类似，在画笔模式下，颜料会被涂抹到物体表面，而在橡皮擦模式下，则会擦除表面上的颜料。

接下来是画笔参数。与传统的绘画软件不同，为了做出比较逼真的效果，绘制到物体表面的除了颜色之外，还有一些表示表面特性的属性，例如反射率，光泽度，表面厚度等等。根据这些属性，就可以模拟出一些比较常见的表面效果。

在画笔参数栏中，首先有两大选项，一个是“使用自带颜色”，而另一个是“使用颜料拾取器”。若选择使用自带颜色，绘制过程与通常的绘画软件差不多。唯一的不同是除了颜色之外，还可以调节各种材质参数。下面列出各个参数的意义：



图4-8 颜料面板

颜色：控制物体表面的颜色

流量：控制绘制的颜料多少，实际上是表面的不透明度

反射：控制表面反射率

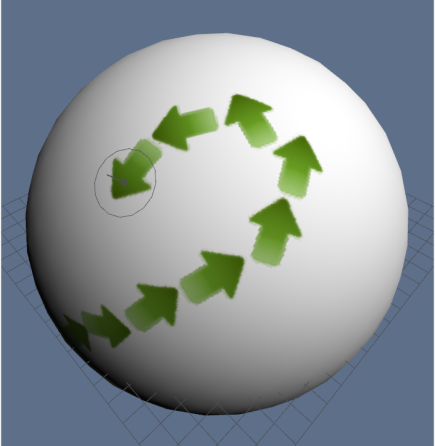
光泽度：控制表面的光滑程度，较高的光泽度会使得反射影像变得清晰，同时高光范围变小

折射：控制物体表面的折射，在当前的实现中，这个属性暂时是无效的。

金属度：这个属性本来称为折射率（IOR），是菲涅尔反射模型的一个参数，可是由于这个属性实际上起到调节物体金属质感强弱的作用，因而称为金属度。

厚度：表面的厚度，利用这个参数，可使得表面看起来凹下去或者凸出来。

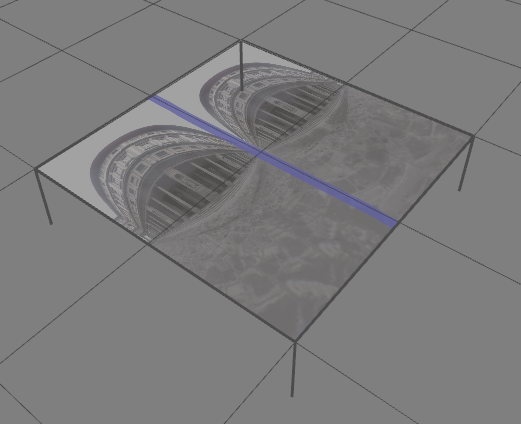
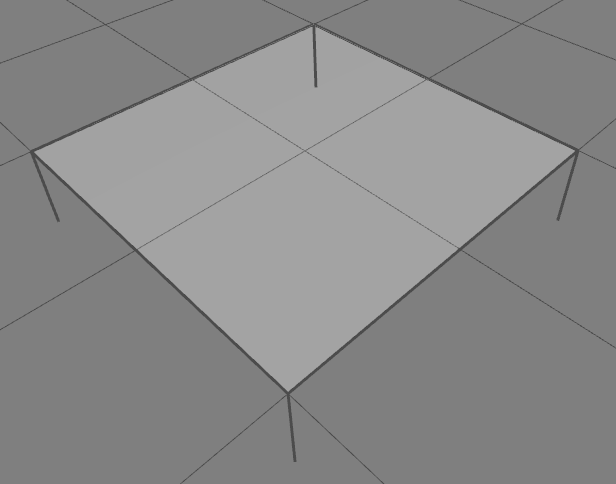
每种属性后面都有个勾选框，这个框代表对应的属性是否发生作用。而在调节属性值的滑块之后，有两个按钮，一个带绿色加号，另一个带红色减号。点击加号，可以选择一幅图片，之后图片会在加号左方显示，如图4-9 a。在绘画的时候，画笔会把这张图片反复印到物体表面。如图4-9b。



a)选择图片作为颜色通道值 b)实际效果

图4-9

若选择使用颜料拾取器，则需要预先定义一个颜料拾取器。当前我们只实现了平面的颜料拾取器。图4-10a显示了刚创建的颜料拾取器，拾取器是一个平面，四个角的四条射线表示拾取器投射像素的方向。实际上，颜料拾取器本身也是一个可以绘制的对象，创建完了颜料拾取器之后，可以用画笔在上面绘制，也可以导入一幅图片到颜料拾取器（导入图片的方法会在后面提及），导入之后如图4-10b所示。



a) 创建后 (b)加载图像后

图4-10 颜料拾取器

这时候，就可以把颜料拾取器的颜色投射到物体表面了。方法是把拾取器移动到物体的周围，然后选中要绘制的物体，点击画笔工具，选择“使用颜料拾取器”，然后点击“从场景中选择拾取器”按钮，这时候鼠标指针会变成一个手势，把鼠标移到物体上单击，可以发现按钮的名字已经变成选中的拾取器名字，这时候就选定了拾取器了。之后在拾取器上移动拖动鼠标，可以发现，移过的地方会自动把拾取器的像素投射到物体表面，如图3-13所示：

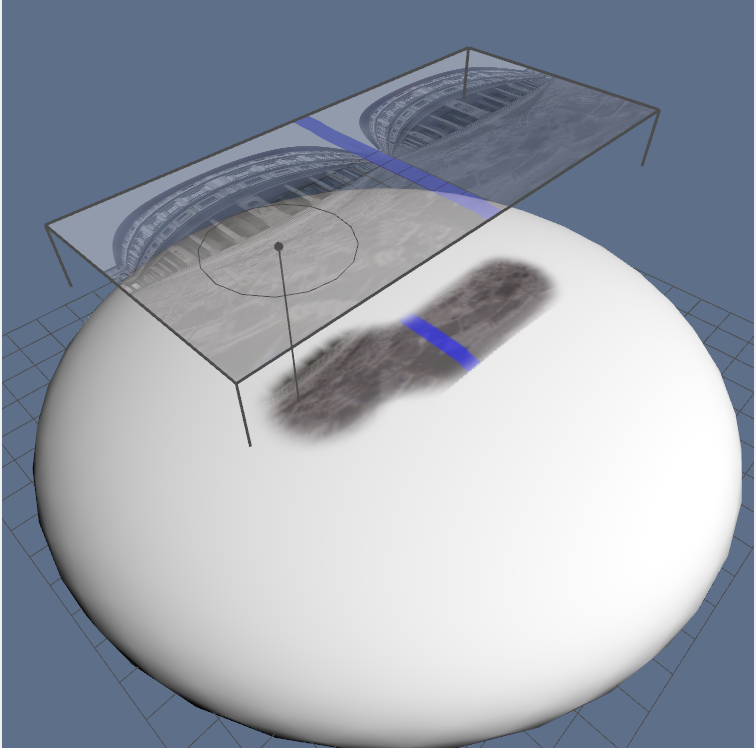
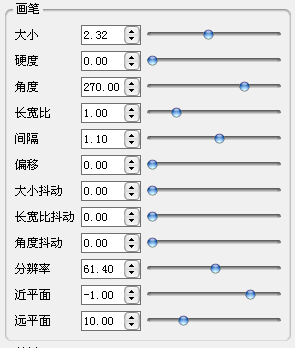


图4-11 利用拾取器在物体表面上绘画

4.3.4 画笔控制

画笔面板负责笔刷参数的控制。图4-12是画笔面板



a) b)

图4-12 画笔面板

画笔面板分成两部分，图4-12a中是画笔的一些参数。这些参数的意义如下：

大小：画笔笔刷的大小

硬度：这个参数仅对默认的笔刷有效，其效果如图4-13所示，图中上侧的笔画硬度为0，而下侧的笔画硬度为0.99.

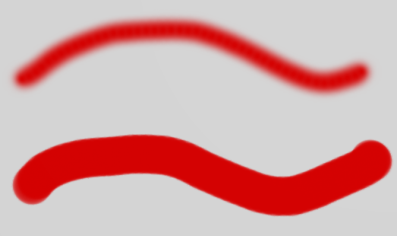


图4-13 不同硬度的笔刷 图4-14 不同角度的笔刷

角度：这个参数仅当使用图片作为笔刷颜色是有用，其效果如图4-14，图中上侧的笔画角度为0度，而下侧的笔画角度为270度.

长宽比：控制笔刷的扁平长度，图4-15是不同长宽比的笔刷效果，上侧笔画的长宽比为1，下侧为0.3：

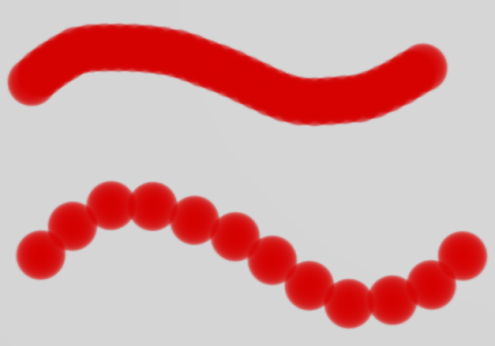
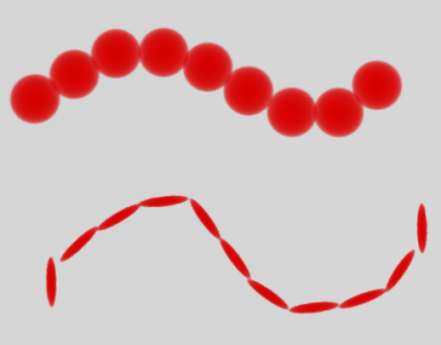


图4-15 不同长宽比的笔刷 图4-16 不同间隔的笔刷

间隔：效果如图4-16，上侧的间距为0.3，下侧为0.8。

偏移：这是一个随机的范围，使得画笔可以在鼠标轨迹周围随机绘制，效果如图4-17。图中上侧笔画的偏移量为0，下侧为0.68。

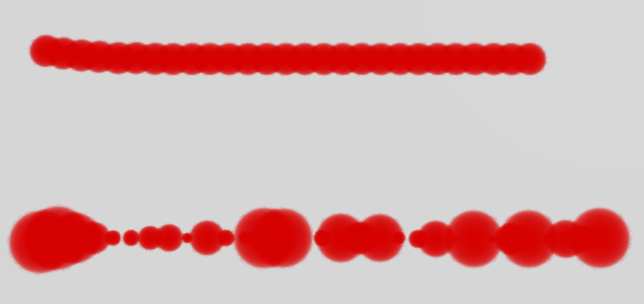


图4-17 不同偏移值的笔刷 图4-18 不同大小抖动值的笔刷

大小抖动：指画笔绘制的过程中，画笔大小的变化范围。效果如图4-18，其中上侧笔画的大小抖动为0，下侧为1。

长宽比抖动、角度抖动与大小抖动类似，就是给各自的属性一个随机值。各自的效果如图4-19和4-20。

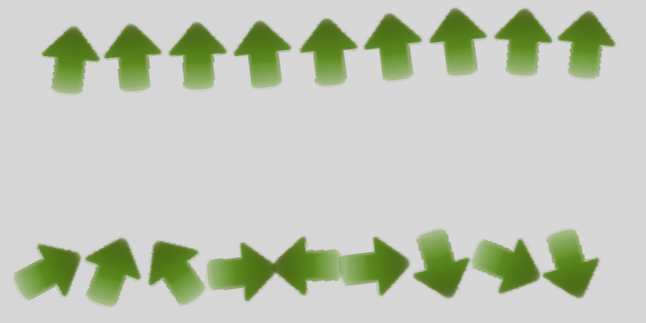
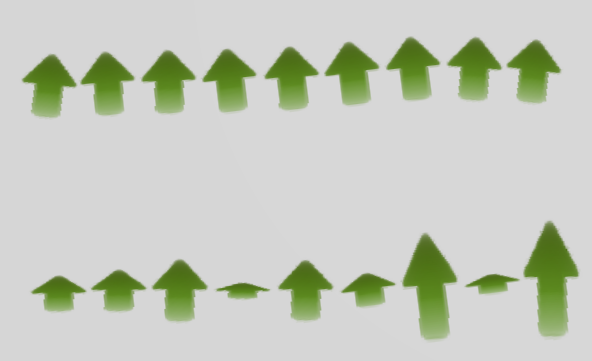
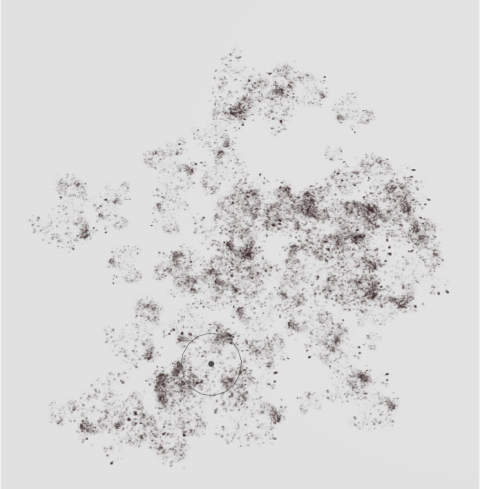


图4-19 不同长宽比抖动的笔刷 图4-20 不同角度抖动值的笔刷

分辨率：如前文所述，在绘制的过程中需要先把笔刷对准的表面的相关信息渲染到一张纹理中。这个分辨率参数就是控制这张纹理的精度的。当笔刷需要对比较大的纹理采样时，就需要把分辨率参数调高，以获得好的采样效果。而调低纹理精度，则可以明显加快笔刷绘制的速度。

近平面、远平面：这两个参数设定了画笔影响的范围。这个范围可以通过三维视图中画笔中心的竖线看出。

图4-12b中是笔触面板，通过把笔触赋予画笔，可以得到比较丰富的绘制效果。图4-21b使用了一幅笔触纹理来做出类似污渍的效果。



a) 所选择的笔触纹理 b) 实际效果

图4-21 使用笔触纹理

在系统中，既可以读取图片作为笔触，也可以读取photoshop的abr笔刷库文件，利用里面的笔刷预设。网上有很多这样的笔刷库，用户可以轻而易举地利用这些资源创作出满意的效果。

4.3.5 物体变换

如图3-24所示，这个面板显示物体的名称，以及平移、旋转、缩放等参数。它们都可以被用户修改。值得注意的是，在系统中不允许有名字相同的物体，一旦发现物体的新名字跟现有物体名字冲突，系统会自动在名字后添加后缀，直到不冲突为止。

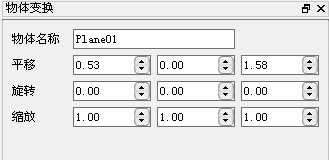


图4-22 物体变换面板

4.3.6 历史记录

与大多是绘图程序一样，本系统支持历史记录。历史记录面板专门记录了用户最近进行的操作，如图3-25所示。用户既可以通过工具栏上的撤销/重做按钮撤销或重做命令，也可以通过快捷键Ctrl+Z或者Ctrl+Y来完成同样的事情。如果用户想要撤销或重做多个操作，可以在命令列表上直接点击要退回或前进到的步骤。



图4-23 历史记录面板

4.3.7 材质层

如图3-25是材质层面板，其功能类似于Photoshop的图层窗口。主要是为了管理物体表面的各层材质。由于时间所限，只实现了最基本的编辑功能。包括增删材质层，层与层之间的顺序调整，读入外部文件到层的相应属性通道中，以及把选中的属性通道输出成图片。

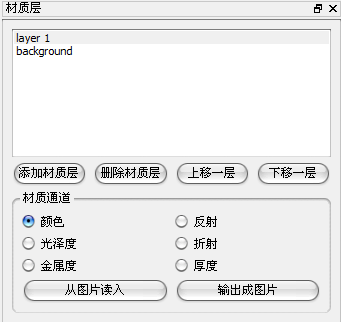


图4-24 材质层面板

在图中列表中选中要修改的层，画笔绘制时，就会修改选中的层的像素值，而不会影响到其他层。

4.4系统功能测试

下面是对系统功能的测试。

图4-25是启动程序之后的界面。

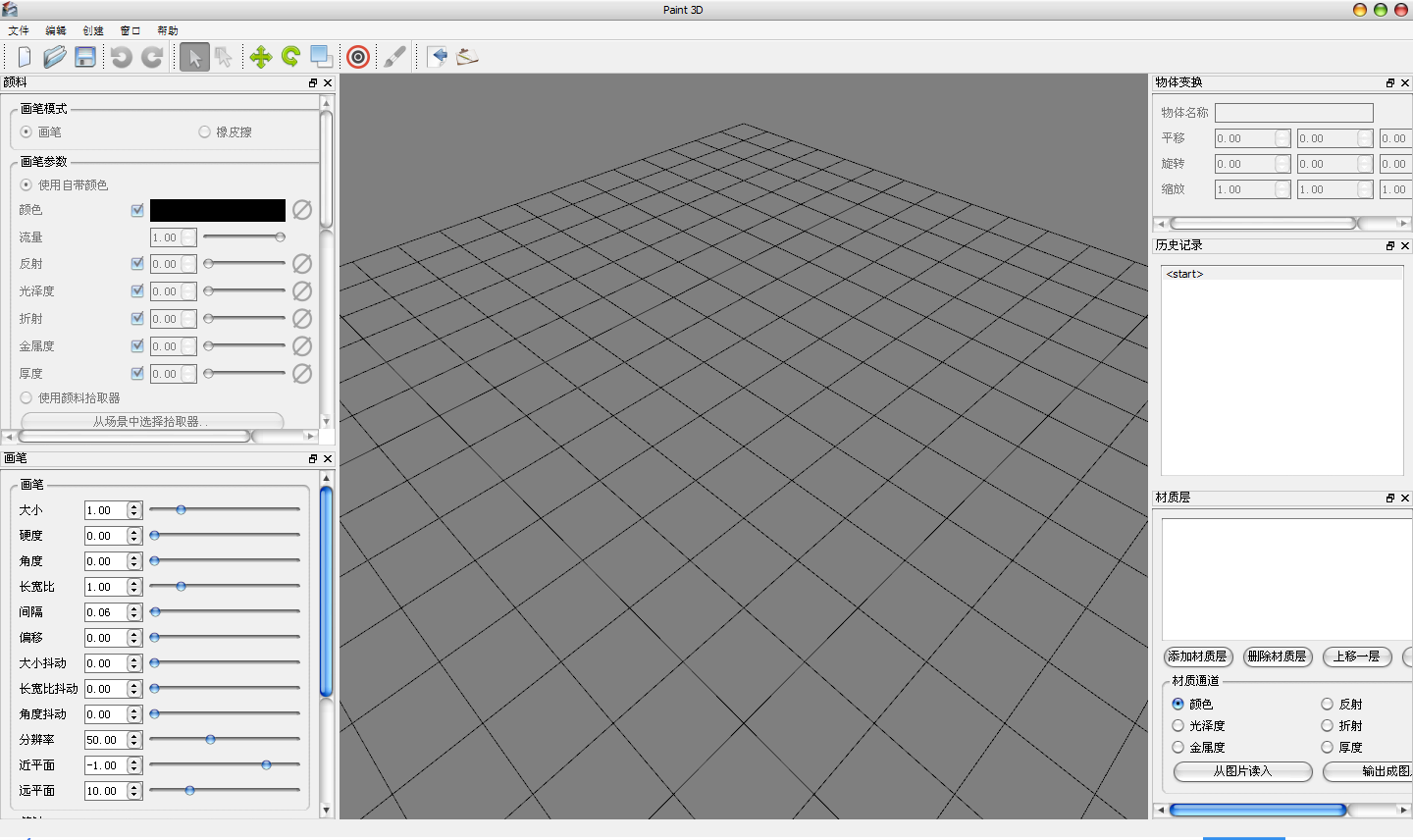


图4-25 程序启动界面

启动程序后，首先需要导入要进行绘制的模型。导入后如图4-26所示：

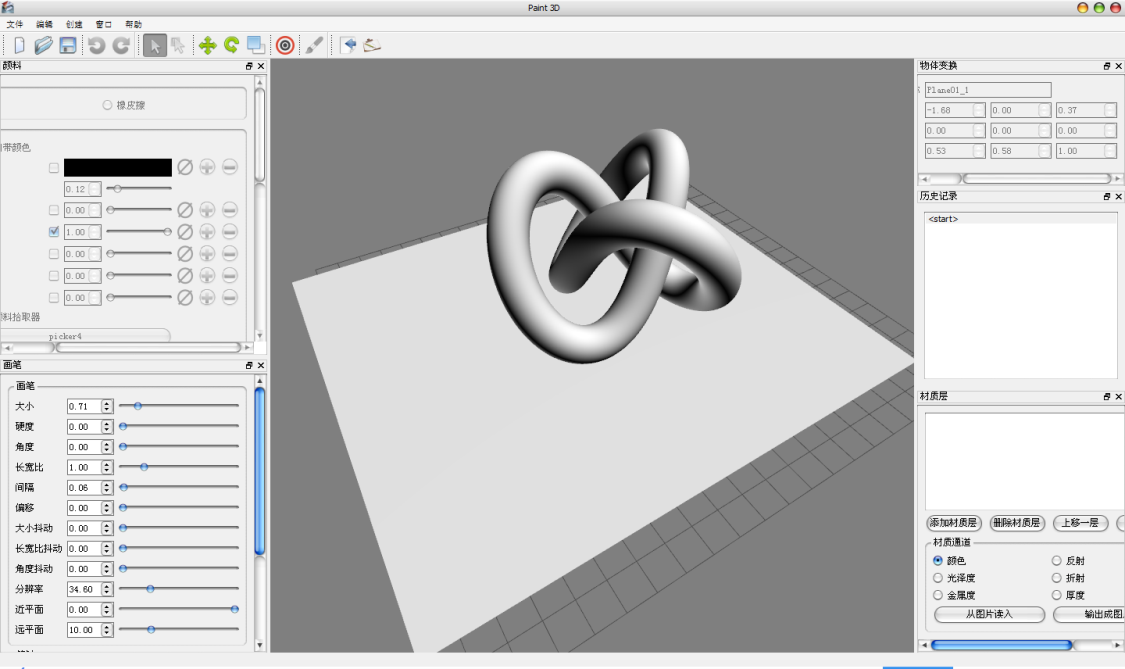
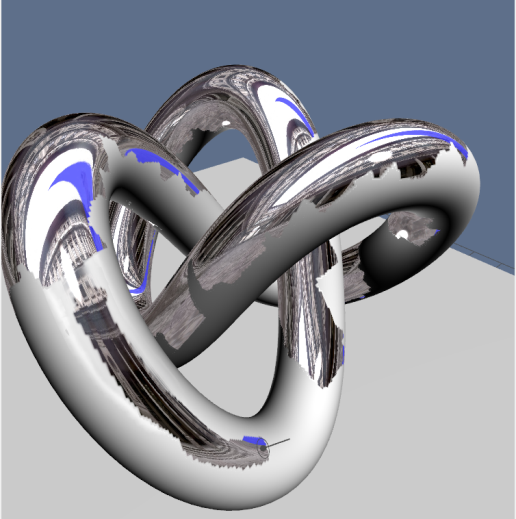
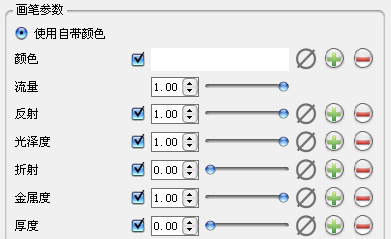


图4-26 导入模型后的程序界面

模型导入后，首先对其中的扭结物体进行绘制。为了得到类似金属的质感，需要在颜料参数面板中把反射属性调到最高，同时把光泽度调为1，以模拟完全光滑的表面。最后，还要把金属度属性调为1，以得到金属质感。图4-27a为相应的颜料参数设置，图4-27b则 为在物体上绘制的情形。



a) 参数设置 b) 在物体表面上绘制

图4-27 绘制金属表面

下一步，对平面进行绘制，使它具有破旧大理石的外观。首先，给平面上褐色底色，为了在随后的步骤中表现表面的裂缝，需要把颜料的厚度属性调高。绘制后的表面如图4-28。



图4-28 往地面添加底色 图4-29 读入abr笔刷

在平面上绘制裂纹。为此先载入笔刷纹理，再从中选择合适的裂纹笔触，如图4-29。然后，为了增加裂纹的随机性，增大笔刷的各项抖动参数。绘制后的裂缝如图4-31。可见，此时，平面已经具有凹凸质感。

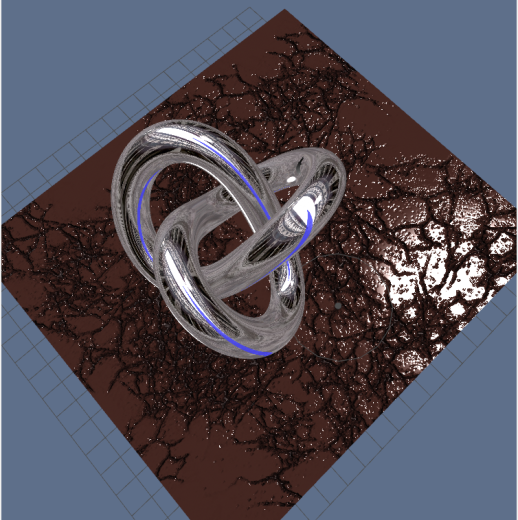


图4-31绘制的裂缝 图4-32 添加材质层

接下来绘制喷溅到扭结和平面上的白色油漆。为了不影响之前绘制的结果，需要分别往两个物体添加新的材质层，如图4-32。然后，选择合适的画笔笔触，在新的材质层上绘制油漆喷溅的效果。

使用类似的办法，往扭结和地面添加污渍。

最终效果如图4-33所示。从图中可见，系统已经能够比较好地模拟出物体的表面质感。此时如用户对编辑结果满意，可以把各层的纹理保存为图片，导入三维软件的材质编辑系统中使用，同时把当前的绘制结果保存为一个q3d文件。

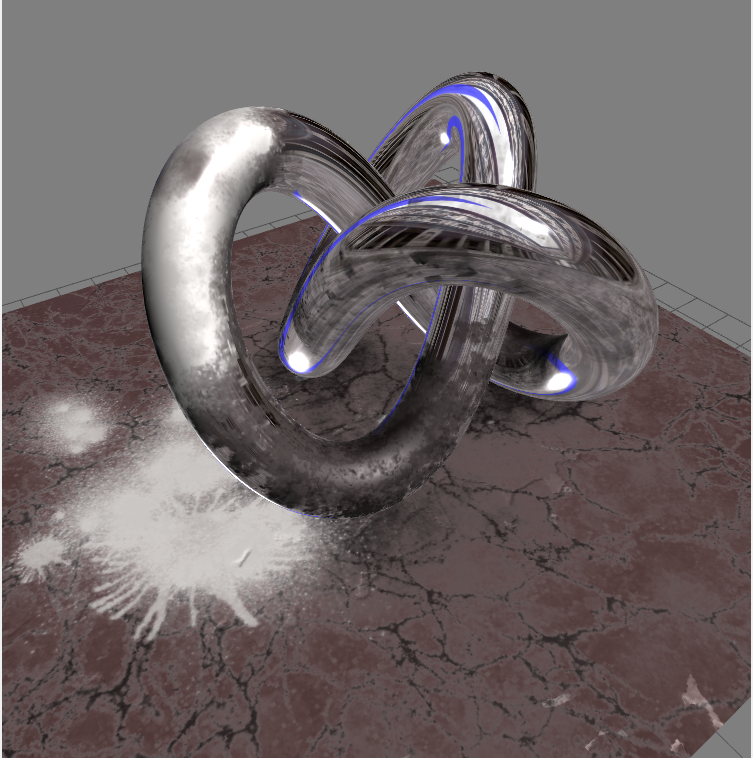


图4-33 最终效果

上述测试表明，系统能够比较快捷地表现出物体表明的质感。

4.5 本章小结

本章系统介绍了系统的实现、系统的各个主要功能模块以及对系统的功能测试结果。系统采用Qt界面库，其强大的界面构建功能，以及齐全的图形库，还有详尽的帮助文档极大地提高了编程效率。系统集成了进行纹理绘制的基本功能，利用这些功能用户能够绘制出表面的丰富细节，快速地表现出物体的质感。

第五章 结论

本文介绍了一种新的材质纹理编辑方法，以及使用这种方法所实现的一个材质纹理编辑系统。当前，已经有一些直接在三维物体上绘画的软件，如Bodypaint3D、Mari、Modo等，相比于先前只能在物体的纹理坐标空间中编辑纹理贴图，这些软件为用户提供了极大的方便。但是，由于物体表面的各种质感属性往往是互相关联的，只能单独影响一个属性的做法无法准确地表现物体表面细节。而相比于传统的编辑方法，新的方法用少数的几个额外参数模拟物体表面反射率、光泽度等表面属性，同时提供能够同时编辑这些属性的方法，在编辑的时候，可以根据物体表面属性自动调整绘制的结果，因此能够更快捷地表现物体表面的质感。

本系统的应用主要在3D艺术领域，如影视特效创作、游戏制作等。在影视特效制作领域，常常需要逼真地表现物体表面的质感细节；而在游戏制作领域，当前流行的所谓次世代游戏制作，则希望更好地表现物体表面的凹凸细节。因此，在这些领域中，是存在广泛的需求的。

总的来说，本系统基本达到了预定目标，利用本软件，能够表现物体表面的一些质感。在绘制完成后，可以把各层的每个通道分别保存为图片，导入三维软件中使用。

但是，软件功能尚未完善。主要有以下几点：

1. 还没有实现油漆桶功能，当需要修改大片面积的颜色是，只能够用画笔慢慢刷。
2. 当前图层的分辨率被限制为1024x1024，不能调节，这降低了使用的灵活性，很多时候消耗了过多的系统资源。
3. 对于纹理接缝的问题，暂时还不能很好地解决。在当前的算法下，要完全消除接缝是不可能的，只能够在一定程度上减轻接缝处的瑕疵，得到如图2-13的效果。如果要完全消除纹理接缝，必须采用新的算法。
4. 在绘画的过程中，当笔刷比较大时，效率仍然比较低，用户体验不佳。
5. 当前系统没有提供物体纹理坐标空间图像的编辑功能。在有些情况下，直接在纹理坐标空间中编辑要方便得多。而这个编辑过程更加类似传统的二维绘画，如要使得在二维编辑变得可行，还必须实现一系列常用的工具，例如画线、画圆、选区工具等。

在下一步的实现中，需要进一步完善软件的功能，同时对于算法上的缺陷，需要寻找相应的解决方法。这些工作如果一个人做，往往是十分艰难的，因此，还应该积极寻找感兴趣的合作者，一起把系统完善。

参考文献

1. Maneesh Agrawala, Andrew C. Beers, Marc Levoy. 3D Painting on Scanned Surfaces〔J〕.  Symposium on Interactive 3D graphics ([SI3D) 1995:145-150, 215](http://www.dblp.org/db/conf/si3d/si3d95.html" \l "AgrawalaBL95)
2. George ElKoura. A 3D Interactive Texture Painter〔EB/OL〕. http://www.dgp.toronto.edu/~gelkoura/texpaint.pdf
3. Bart Adams, Martin Wicke, Philip Dutré, Markus Gross, Mark Pauly, Matthias Teschner. Interactive 3D Painting on Point-Sampled Objects〔EB/OL〕, http://graphics.cs.kuleuven.be/publications/I3DPoPSO/
4. Nelson Chu, William Baxter, Li-Yi Wei, Naga Govindaraju. Detail-Preserving Paint Modeling for 3D Brushes〔J〕. [Non-photorealistic Animation and Rendering](http://www.npar.org/" \t "_blank)([NPAR) 2010:27-34](http://www.dblp.org/db/conf/npar/npar2010.html" \l "ChuBWG10)
5. Bill Baxter, Vincent Scheib, Ming C. Lin, Dinesh Manocha. DAB: Interactive Haptic Painting with 3D Virtual Brushes〔J〕，[SIGGRAPH 2001:461-468](http://www.dblp.org/db/conf/siggraph/siggraph2001.html" \l "BaxterSLM01)
6. Yanyun Chen, Lin Xia , Tien-Tsin Wong, Xin Tong, Hujun Bao, Baining Guo, Heung-Yeung Shum. Visual Simulation of Weathering By γ-ton Tracing〔J〕，[ACM Trans. Graph. (TOG) 2005,24(3):1127-1133](http://www.dblp.org/db/journals/tog/tog24.html" \l "ChenXWTBGS05)
7. Stephane Merillou, Jean-Michel Dischler, Djamchid Ghazanfarpour. Corrosion: Simulating and Rendering〔J〕，[Graphics Interface 2001:167-174](http://www.dblp.org/db/conf/graphicsinterface/graphicsinterface2001.html" \l "MerillouDG01)
8. Julie Dorsey, Pat Hanrahan. Modeling and Rendering of Metallic Patinas〔J〕，[SIGGRAPH 1996:387-396](http://www.dblp.org/db/conf/siggraph/siggraph1996.html" \l "DorseyH96)
9. Eric Paquette , Pierre Poulin, George Drettakis. The Simulation of Paint Cracking and Peeling〔J〕. [Graphics Interface, 2002:59-68](http://www.dblp.org/db/conf/graphicsinterface/graphicsinterface2002.html" \l "PaquettePD02)
10. Brent Burley, Dylan Lacewell. Ptex: Per-Face Texture Mapping for Production Rendering〔J〕. [Computer Graphics Forum (CGF) ,2008,27(4):1155-1164](http://www.dblp.org/db/journals/cgf/cgf27.html" \l "BurleyL08)
11. Budirijanto Purnomo, Jonathan D. Cohen and Subodh Kumar. Seamless Texture Atlases〔J〕, [Symposium on Geometry Processing 2004:67-76](http://www.dblp.org/db/conf/sgp/sgp2004.html" \l "PurnomoCK04)
12. [Francisco González](http://dl.acm.org/author_page.cfm?id=81405594549&coll=DL&dl=ACM&trk=0&cfid=104735606&cftoken=22383228" \t "_self" \o "Author Profile Page), [Gustavo Patow](http://dl.acm.org/author_page.cfm?id=81100593424&coll=DL&dl=ACM&trk=0&cfid=104735606&cftoken=22383228" \t "_self" \o "Author Profile Page). Continuity mapping for multi-chart textures〔J〕, ACM Transactions on Graphics (TOG) , 2009, Volume 28 Issue 5 ,Article No. 109
13. Michael A.Lachance, Arthur J. Schwartz, Four point parabolic interpolation〔J〕. Computer Aided Geometric Design,1991, 8: 143-149
14. M. A. Kaplan, R. A. Papetti, A Note on Quadrilateral Interpolation〔J〕. Journal of the Association for Computing Machinery, October 1971,Vol. 18, No.4, pp. 576-585
15. [Heinrich Dorrie](http://www.amazon.com/s/ref=ntt_athr_dp_sr_1?_encoding=UTF8&field-author=Heinrich%20Dorrie&search-alias=books&sort=relevancerank), 100 Great Problems of Elementary Mathematics〔M〕, 2nd edition,  New York: Dover Publications, June 1, 1965

致谢

在系统的实现过程中，我的指导老师给了我很多有益的建议，使我受益匪浅，在这里表示诚挚的感谢。还要感谢我的室友，他们为我营造了安静的学习环境。

另外，在代码的编写过程中，Qt完备的文档使我能够快速地找到各种问题的答案，极大地提高了开发效率。因此，在这里衷心地感谢Qt的研发人员，祝他们工作顺利，身体健康，同时祝愿Qt越来越受欢迎。