哈 尔 滨 理 工 大 学

毕 业 设 计

题　　目：基于手绘草图的三维模型检索

院、　系：计算机科学与技术

姓　　名：邱泳锋

指导教师：高雪瑶

系 主 任：孙东璞

2021 年 6 月 13 日

基于手绘草图的三维模型检索

摘　要

随着计算机 CPU 和 GPU 的运算能力的大幅提升，三维模型不但变得越来越复杂，细节信息越来越丰富，并且在在动画、机械、医疗等领域应用更加广泛，三维模型数量也越来越多，三维模型的分类与检索已经成为一个重要的研究方向。尽管现在有很多学者对三维模型检索技术进行了各种不同的研究，并且提出了许多不同的模型检索算法，但目前仍有许多问题亟待攻克。

本文通过对现有的三维模型检索技术进行研究和分析，发现三维模型内在的复杂性和高维度的计算量严重影响了模型的检索。本文采用三维模型降维的方式，利用二维视图作为三维模型的检索条件以此来降低检索成本。提出一种基于手绘草图的三维模型检索，本文主要研究了以下内容。

为了提高检索的准确性，首先对三维模型进行合适的空间位置变化，进行渲染，然后按照固定投影的方法获取二维视图集。每一个模型选取 6 幅二维视图作为三维模型的特征视图集。然后对手绘草图和二维视图集提取特征向量，以 Zernike 矩和 Fourier 描述符来构建全局视图特征和 D2 描述符的加权集合作为集成特征描述子。通过集成特征描述子来充当用户手绘草图和模型的二维视图的特征向量，以此进行相似性评估来检索三维模型。实验结果表明该方法能够有效地对三维模型进行分类。

关键词　三维模型检索；手绘草图检索；Zernike 矩；Fourier 描述符；D2 形状描述符；集成特征描述子

3D model retrieval based on hand drawn sketch

Abstract

With the rapid development of CPU and GPU, 3D models are not only becoming more and more complex and rich in details, but also widely used in animation, machinery, medical and other fields. The number of 3D models is also increasing. The classification and retrieval of 3D models has become an important research direction. Although many scholars have done different researches on 3D model retrieval technology and proposed many different model retrieval algorithms, there are still many problems to be solved.

In this paper, through the research and analysis of the existing 3D model retrieval technology, it is found that the inherent complexity of 3D model and the high-dimensional computation seriously affect the model retrieval. In this paper, 3D model dimension reduction method is adopted, and 2D view is used as the retrieval condition of 3D model, so as to reduce the retrieval cost. This paper proposes a 3D model retrieval method based on hand drawn sketch.

In order to improve the accuracy of retrieval, firstly, the 3D model is rendered according to the appropriate spatial position change, and then the 2D view set is obtained according to the fixed projection method. Each model selects 6 2D views as the feature view set of 3D model. Secondly, feature vectors are extracted from sketch and 2D view sets, and weighted sets of global view features and D2 descriptors are constructed by Zernike moments and Fourier descriptors as integrated feature descriptors. By integrating feature descriptors to serve as feature vectors of user's hand drawn sketches and 2D views of the model, similarity evaluation is performed to retrieve 3D models. Experimental results show that this method can effectively classify 3D models.

Keywords 3D model retrieval, Freehand sketch retrieval, Zernike, Fourier descriptor, D2 shape descriptor, Integrated feature descriptor

目　录

第1章 绪论

1.1 课题背景

随着三维建模相关技术的进步尤其是三维扫描设备的成熟及大众化推广，互联网空间已经积累了大量可分享的三维模型，且模型数量呈持续剧增之势。譬如，用户可以直接从谷歌三维仓库(3D Warehouse)下载海量三维模型;著名的 3D 模型交易平台 TurboSquid 目前已拥有超过 30 万个模型。为充分利用已有的三维模型，需要发展便捷、高效、可靠的三维形状(模型)检索引擎。三维形状检索即通过特定的交互操作从数据库中查找出符合用户意向的三维形状。其中，检索的对象可以是整个模型(如一张椅子)，也可以是模型的一部分(如椅子的把手)。

三维模型不仅在数量方面快速增长，而且应用领域也越来越广泛。在工业产品设计、建筑设计、虚拟现实、计算机仿真、多媒体教学、分子生物学、教育、三维游戏和影视动画中都广泛地使用了三维模型，三维数据模型库也已经越来越普遍。在互联网上，也涌现出越来越多的三维模型库。三维模型已经成为 MPEG-7 标准模型的一个重要研究部分。因此，三维模型的检索是当今图形检索中的一大热点

1.2 研究意义

相关的三维建模企业要想在当前激烈的市场竞争中优先抓住商机，生产出满足用户需求的产品，从中获取利润，就不得不思索降低三维模型设计和三维模型加工的高额成本。研究调查表明：在产品设计过程中，只有20%左右的设计需要从头作起；有 40%的设计可以通过直接重用目前已有的设计来完成；另外 40%的设计可以对已有设计作适当修改而得到。如果每次生产新产品时，都从头选材，绘制图纸，设计加工方案，不仅耗时而且也增加了成本。在生产过程中，经常会出现重复设计的现象，其本质原因是没有高效的 CAD 模型检索工具。快速而又有效的 CAD 模型检索无疑会大大缩短开发周期，不仅能提高设计效率，而且能为企业带来更多的利益。重用已有的 CAD 模型也会让企业节约更多的成本。

随着三维 CAD 模型的数量和种类急剧增加，CAD 模型库的规模也随之变大。据统计目前全世界已有 300 亿个三维模型。三维模型结构复杂种类繁多，如何能够精确地从模型库中寻找满足用户设计意图的模型已变得越来越重要了。因此，三维 CAD 模型检索技术正受到国内外专家和学者

的广泛关注。

三维 CAD 模型检索是计算图形学和计算机辅助设计(Computer Aided Design，CAD)领域中的一个重要研究课题，也是近年来这些领域的一个研究热点。该研究不仅对三维 CAD 模型重用具有很高的实用价值，而且也大大地加深了人们对计算机辅助设计和人工智能等问题的理解，促进相关学科的发展。

1.3 国内外研究现状

近年来，国内的对图形检索的研究也有了更进一步的发展，目前国内对于三维图形检索的研究中，关于草图识别方法主要包括以下几类：

(1)基于收集统计数据和资料的识别方法，通过对草图中的各类线型进行数理统计，并依靠统计得到的信息进行识别。

(2)基于模糊类的识别方法，运用模糊处理相关原理和技术，对于草图位置、笔划速度等绘制特征参数来进行识别。

(3) 基于交互式几何帧识别方法，分析降噪草图的笔画特征，然后通过笔画中反映的几何特征找出相似的因变量，将笔画形成的角度信息与预设阈值进行比较分类，实现草图识别。

(4) 基于滤波器和神经网络的识别方法，即构造滤波器对笔画进行分类。

基于手绘草图的三维模型检索，虽然不同的使用者会根据自己的主观意向来对同一个模型进行不同的描述，但是一个模型的组成是固定不变的。举例而言，一个显示器由两个主要的部分组成，一个是底座，另一个是屏幕。绝大多数的屏幕呈类矩形的，屏幕必须要在底座上方，显示器屏幕远远大于显示器的底座。由此可以断定，我们可以根据使用者绘制的不同的简易草图，便可以简单的确定出用户的欲检索的期待目标。Sezgin T M 等[1]提出了一种能够识别出线，圆等几何图形的算法，Li B 等[2]开发了一种基于监督学习的正能草图识别器，可以正确的获得草图的所表达出的语义。Juefei[3]建立了基本的基于草图的三维场景检索基准，并在该基准上评估了 14 种基于草图检索的检索方法。Zhu[4]将三维图形投影到二维空间，并使用自动编码器对二维图像进行特征学习。用深度学习特征对传统局部图像描述子进行补充。Konstantinos 等[5]和周燕等[6]利用三维模型的二维全景表示作为卷积神经网络的输入，利用卷积神经网络来计算特征。张云峰[7]利用图像和三维模型表述信息的互补性来建立图像与模型的关系。张艺琨等[8]在 Canny 边缘信息的基础上,进一步提取形状上下文特征描述全局信息,融合 ORB 特征和形状上下文特征得到一个新的特征表示三维模型。李海生等[9]提出一种基于模型内二面角分布直方图的特征描述方法。安勃

卿[10]则利用深度学习技术解决基于手绘草图的三维模型检索。

1.4 课题的主要研究内容

对三维模型检索策略进行研究。在分析二维视图特征定义和三维模型相似性计算的基础上，探索一套基于草图的三维模型检索方法。从不同视角对三维模型投影获取对应的二维视图。计算草图与三维模型每一个二维视图之间的相似性。选择最大相似性数值作为草图与三维模型之间的相似性。同时，开发一个基于草图的三维模型检索系统。

通过阅读大量的文献，对国内外三维模型检索方法的起源、发展和趋势有 了深刻的理解，分析并总结目前所面临的问题与不足，对三维模型的检索方法 进行了仔细分析和深入研究，本文主要研究内容为以下几个部分:

1)从不同的视角对三维模型投影获取对应的二维视图，搜集所有的二维视图形成最优视图集合。

2)综合利用多种特征来描述草图和三维模型的二维视图，使用欧氏距离来计算草图和二维视图之间的相似性。基于三维模型的二维视图集计算草图 与三维模型之间的相似度。

3)开发一个三维模型检索系统，从模型库中找出与草图最相似的三维模型，并显示多个相似的三维模型排序列表。

1.5 本文文章结构安排

对三维模型检索策略进行研究。在分析二维视图特征定义和三维模型相似性计算的基础上，探索一套基于草图的三维模型检索方法。通过不同视角对三维模型投影获取对应的二维视图。计算草图和三维模型的每个二维视图之间的相似性。选择最大相似度值作为草图和三维模型之间的相似度。 同时，开发了基于草图的三维模型检索系统。论文共 6 章，各章的内部结构安排如下:

第 1 章，绪论。本章首先简要介绍了关于三维模型检索的研究背景及意义，并论述了目前基于二维视图的三维模型在国内外的研究现状，以及目前存在的问题，然后介绍了本文课题来源及主要研究内容，最后对本文的章节安排进行陈述。

第 2 章，基于手绘草图的三维模型检索的总体框架。该系统主要包括两大部分，一个是模型投影的处理，另一个是检索系统的处理。针对这两大部分进流程梳理与介绍。

第 3 章，三维模型的处理。本章主要介绍了本文实验所需的数据集合，解析了使用的模型格式，具体介绍了各种投影方式坐标空间的变化以及光照效果的添加。将模型进行渲染，得到合适的三维模型，并进行投影得到二维视图集。

第 4 章，草图的绘制处理。本章主要介绍了手绘草图画板的设计。在硬渲染的基础上，采用软渲染的算法，实现了基本图形的绘制，包括直线，矩形，三角形，圆形，曲线和铅笔线的绘制。

第 5 章，草图及三维模型的特征提取。本章提出由全局视图描述符包括 Zernike 矩和 Fourier 描述符和二维形状分布结合的集成描述子，用于提取手绘草图与二维视图的特征，以此来解决单一描述符对二维视图特征提取不全面的问题。

第 6 章，基于手绘草图的三维模型检索。本章主要对比了几种距离计算方法，以此选出一种最好的距离来计算草图与二维视图的相似性。

最后对全文的工作进行总结并对未来发展进行了展望。

第2章 基于手绘草图的三维模型检索的总体框架

2.1 三维模型检索的体系架构

本文实现的检索系统是基于手绘草图的三维模型检索。基于手绘草图的三维模型检索主要包括手绘草图的绘制、手绘草图的特征提取、三维模型的渲染、三维模型的固定投影、投影视图集的特征提取、相似度比较。图 2-1 所示为系统的总体框架。

1.手绘草图的绘制。用户可以自行在在纸张上绘制简易草图，并上传到检索系统中，或者用户可以在系统中给定的简易绘图板中绘制草图，绘图板上集合了绘制草图的部分所需的工具(直线，矩形，三角形，圆形，曲线，铅笔线)。

2.手绘草图的特征提取。本文提出了由全局视图特征和 D2 描述符结合的集成描述符作为新的描述符，全局视图特征是 Zernike 矩和Fourier 描述符结合而成，用此描述符提取手绘草图的特征检索三维模型。

3.三维模型的渲染。本系统的三维模型采用的是 Net40。该模型库中含有 40 类不同的三维模型。并且该模型的存储文件格式为 OFF 文件格式。因此有一个简易渲染器可以渲染出所有模型，并且该渲染器集成了一些基础操作(模型的旋转，缩放，移动，光照材质的添加以及修改)。

4. 三维模型的固定投影。如果直接使用二维草图来进行三维模型检索，由于三维模型的尺寸问题会导致特征相差较大，本文的方法是将现有的三维模型进行一定的处理如模型降维操作，即通过投影的方式将三维模型变成二维视图的表示。

5. 投影视图集的特征提取。采用和提取草图特征一样的方法提取模型的特征。

6. 相似度比较。用距离公式计算出与草图最相近的 8 个检索模型。

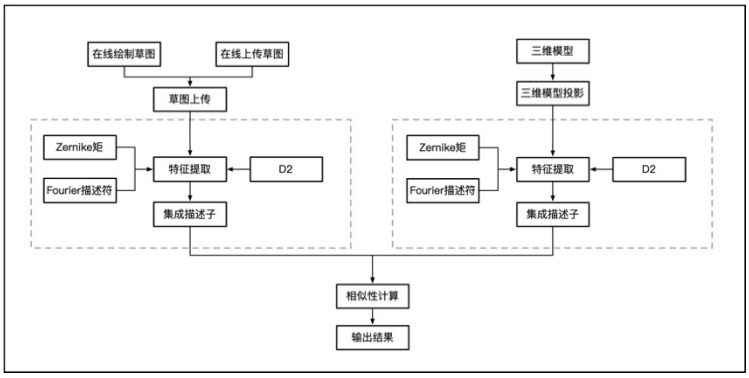

图 2-1 系统的总体框架

2.2 三维模型检索的处理流程

2.2.1 三维模型投影的处理架构

基于手绘草图的三维模投影过程如图 2-2 所示。先对 OFF 格式的模型进行解析，将模型导入系统后，给模型添加材质和各种光照效果，接着对模型进行平移，旋转，缩放。在渲染出的三维模型中，按照一定的角度，进行投影。

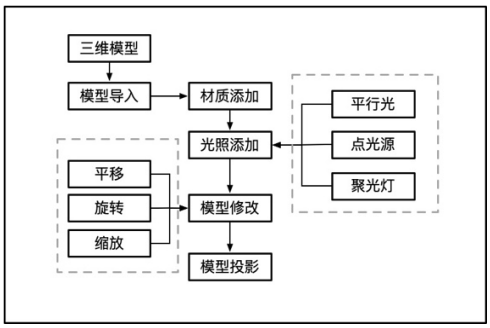

图 2-2　三维模型投影的处理架构

2.2.2 三维模型检索的处理架构

基于手绘草图的三维模型检索过程分为在线检索与线下检索两个阶段，如图 2-3 所示。

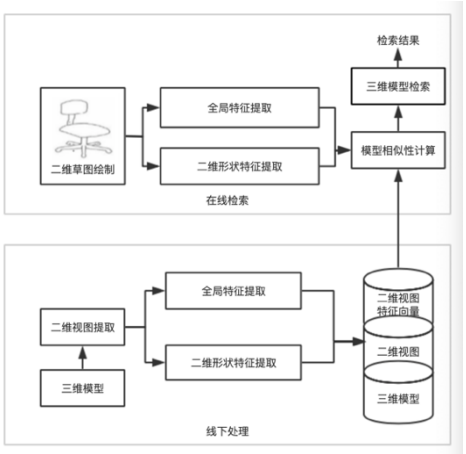

图 2-3　三维模型检索的处理架构

在线下检索过程中，首先利用固定投影的方法对三维模型进行降维处理，获取 6 张三维模型库的二维视图集，最后利用集成描述子提取最优视

图集的全局视图特征与二维形状特征，加权得到 55 维集成描述特征，并存入数据库。在线上用户检索阶段，首先用户进行人工手绘草图，接着利用集成描述子提取集成特征。将相似性与存储在数据库中的三维模型的集成特征进行比较。最后按相似度从大到小依次排序，相似度最高的前 8 个模型显示在检索界面。

2.3 本章小结

本章主要介绍了三维模型检索算法的两部分内容，一部分是三维模型投影的完成流程，包括了模型的渲染，光照，材质的添加以及模型的修改包括了旋转，平移，和缩放。另一部分是完整的检索流程，包括基于草图的三维模型检索流程以分析了各部分模块的功能与作用，给出了流程图。

第3章 三维模型的处理

3.1 OFF 模型文件格式解析及渲染

本系统的三维模型采用的是 ModelNet40。该模型库中含有 40 类不同的三维模型，并且该模型的存储文件格式为 OFF 文件格式。要实现模型的读取、绘制，我们首先需要知道模型是如何存储在文件中的。通常模型是由网格组成的，且一般为三角网格。这是因为其它多边形网格可以容易地剖分为三角形，并且三点共面可以保证平面性以及可以容易地定义内外方向，进行插值等操作。因此，需要先解析 OFF 文件格式。OFF 文件用于表示给定了表面多边形的模型的几何体。这里的多边形可以有任意数量的顶点。Net40 中的 OFF 文件遵循以下标准。OFF 文件全是以 OFF 关键字开始的 ASCII 文件。下一行说明顶点的数量、面片的数量、边的数量。边的数量可以安全地省略。 顶点按每行一个列出 x、y、z 坐标。在顶点列表后，面片按照每行一个列表。对于每个面片，顶点的数量是指定的，接下来是顶点的索引列表。如图 3-1 所示，这是一个浴缸的 OFF 格式模型。

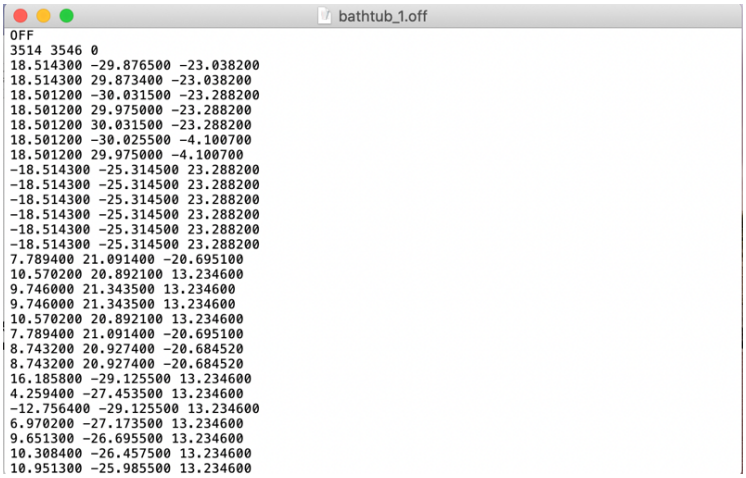

图 3-1 OFF 模型

现在，主流的存储模型的数据结构有面列表，邻接矩阵，以及半边结构。面列表是存储面中顶点的三元组(Vertex1, Vertex2, Vertex3)它的优点是方便而紧凑，可表达非流行网格，缺点是不能有效地支持点、面之间的邻接关系查询。邻接矩阵是表示顶点之间相邻关系的矩阵。它的优点是支持顶点之间的邻接信息 V-V(Vertex to Vertex)的高效查询、支持非流行网格。它的缺点是没有边的显示表达、不支持 V-F(Vertex to Face)，V-E(Vertex to Edge)，E-V(Edge to Vertex)，F-E(Face to Edge)，E-F(Edge to Face)的快速查询。而半边结构是记录所有的面、边和顶点，包括几何信息、拓扑信息、

附属属性，流行于大部分集合建模应用。它的优点是所有查询操作时间复杂度均为 O(1)，所有编辑操作时间复杂度均为)(1)。它的缺点是只能表达流行网格。在该系统中，我采用的是面列表的数据结构。读取模型的流程图如图 3-2 所示。

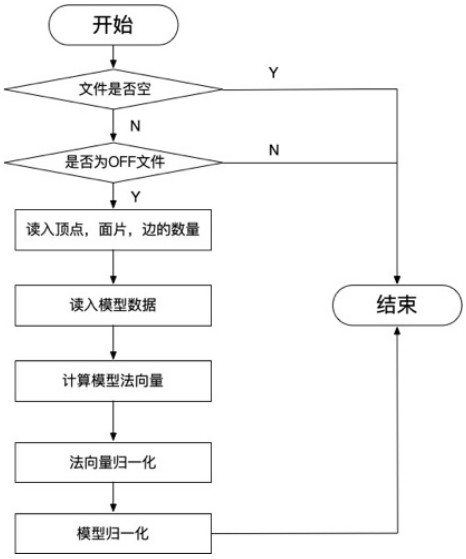

图 3-2 模型流程图

3.2 固定视角的模型投影技术

该系统是是基于手绘草图的检索。目前主流的研究方法一种是获取较为理想的最优视图集合，以此作为检索的数据库，来增加检索的成功率；另一种则是探索出一种较为合适的特征描述符，来当模型的特征向量，以此增加检索的准确率。本文将输入的模型的最优视图定义为 6 张不同的二维视图集。采用固定视角的投影技术。

因为用户所绘制的草图的二维的，而模型是三维的，因此需要将三维模型进行降维处理。避免后续相似性计算时因维数灾难而带来的额外的时间上的开销。此外，不同的用户在绘制二维的草图的时候，事实上已经对三维空间的物体做了一个降维处理。绘制的是一个固定视角的物体的投影。因此，为了适应不同用户可能绘制不同视角下的同一物体。并且考虑到了检索系统的效率，本系统只获取了同一个模型的六个不同视角的投影，并以此组成该模型的最优视图集。在尽可能包含用户所绘制物体的视角的同时，减少因视图集合过多冗余而造成的时间上的额外开销。

为了解决草图和模型在维度上的不统一，Su[11]等人采用了固定视角的方法。该方法的具体过程是，将模型放在水平平面上，使得模型竖直向上。对于每一个充当检索数据库的三维模型，在其水平面以上 30 度角位

置，每隔30度设置一个摄像机，摄像机指向三维模型的中心，每个摄像机获取一张该模型的二维视图。如图所示 3-3。该投影方法可以生成 12 张视图。此外，对于不满足直立向上的模型，将模型周围的正二十面体上的二十个顶点上放置相机。并以此来获取视图。并组成最优视图集。

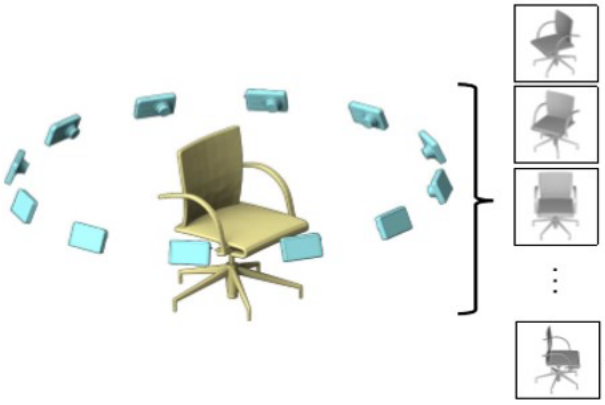

图 3-3 三维模型投影

潘婷[12]提出一种基于球体投影的三维模型检索方法。如图3-4所示用来解决针对域不匹配问题。在对三维模型进行预处理后，在三维模型外围外接一个球体， 在该球体半圆弧上间隔 30° 的地方放置一个摄像机，并确保照相机镜头垂直于模型质心同摄像机之间的连线。在该摄像机每获取一张视图后，将半圆弧旋转 30° 再重复该步骤，直至半圆弧回归原位。如图所示。之后使用高斯差分和贝塞尔曲线完成线图的提取;利用草图和投影图像之间的关系构建分类器，以获取模型的最优视图。

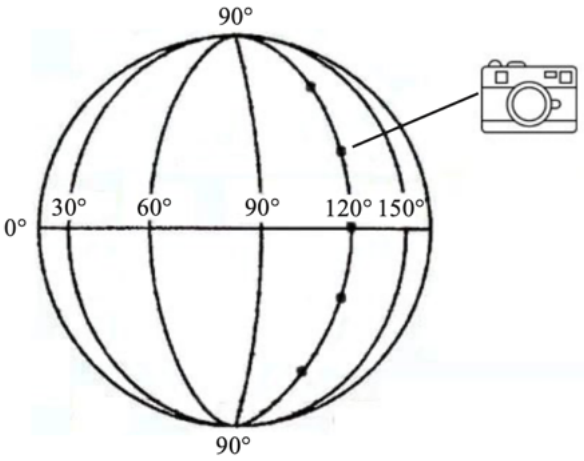

图 3-4 球体投影

Christopher M. Cyr[13]等以规定的间隔（5 度）对观察球进行采样，如图 3-5 所示给出了物体视球空间被划分为若干区域，每个区域对应着一个二维视图。并在迭代过程中，使用曲率匹配和视图相似群组来定义形状

相似度量将视图组合成各个方面。之后将得到的视图集输入数据库中。

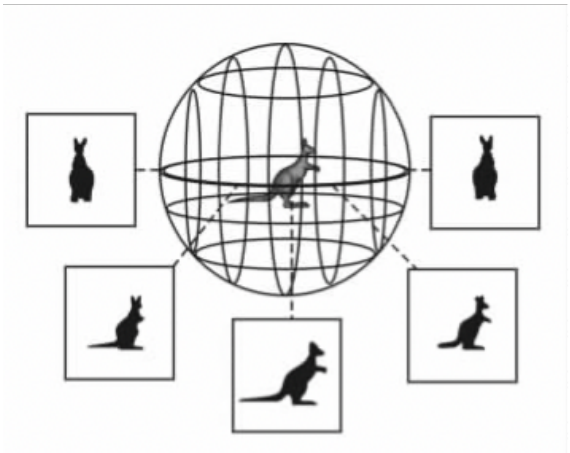

图 3-5 物体视球空间

在进行模型的投影的时候，计算机图形学中，比较常用的两种投影分别是平行投影变化(正交投影)和中心投影变换(透视投影)。如图 3-6 所示。两者之间的区别在于正交投影采用的投影射线之间是相互平行的，而透视投影变换采用的投影射线则是来源于选取的一个公共点，射线之间以椎体的方式投向三维模型。本系统采用的是中心投影即透视投影，因为中心投影更加符合用户绘制草图时的近大远小的绘制方法。

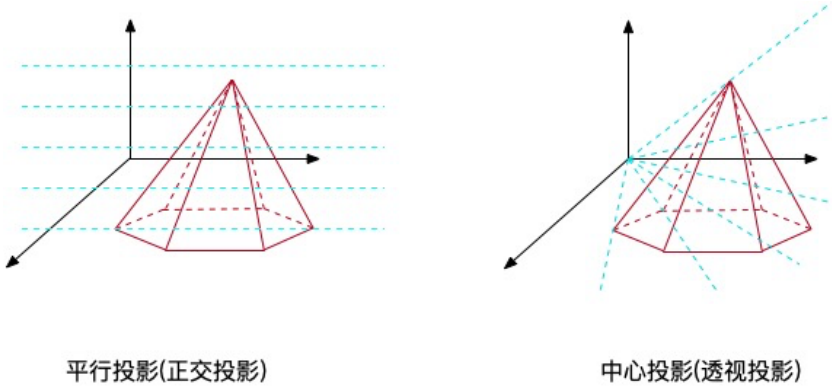

图 3-6 正交投影和透视投影

本文所采用的固定投影的思想是，在 Su 的方法上进行一定的贱货将待投影的三维模型固定于中央，选择固定的角度和固定的投影数量进行投影。对于每一个三维模型，在水平面以上 30 度角位置，每隔 60 度放置一个虚拟摄像机，指向三维模型的中心，该投影方法可以生成 6 张视图。这 6 张视图便组成该模型的最优视图集。

3.3 三维模型的光照和材质添加

本系统中，在读入 OFF 模型后，会自动计算该模型的各个顶点的法向量。使用法向量，系统可以为模型添加照明效果。采用的 Phong 光照模型。Phong 光照模型的主要构建模块包括 3 个组件：环境照明，漫射照明和镜面照明。Phong 光照模型的组成结果如图 3-7 所示。

图 3-7 Phong 光照模型

环境光：光通常不是来自单个光源，而是来自散布在我们周围的许多光源，即使它们不是立即可见的。光的特性之一是，它可以在多个方向上散射和反弹，到达不直接可见的点。因此，光可以在其他表面上反射，并间接影响对象的照明。使用一种小的恒定（浅色）颜色，将其添加到对象碎片的最终最终颜色中，因此即使在没有直接光源的情况下，看起来也总会有一些散射光。物体上一点 P 的环境光光强I_e可以表示为

$$I_e = k_a I_a , k_a \in [0,1] \quad (3\text{-}1)$$

I_a表示来自周围环境的入射光强，k_a为材质的环境反射率。

漫反射光：模拟灯光对象对对象的方向性影响。这是照明模型中视觉上最重要的部分。物体的一部分面对光源的越多，它就会变得越亮。兰伯特余弦定律总结了漫反射光的强度与光入射方向与物体表面法向量之间的角度之间的相关性。当$\theta = 0$时，物体表面正好垂直于光线方向，这是获得的光照强度最大；当$\theta = 90$ 时物体表面与光线方向平行，此时光线照射不到物体，光的强度最弱；最后，物体的表面转向到光线的背面，此时物体对应的表面接受不到光照。物体上一点 P 的漫反射光强I_d表示为：

$$I_d = k_d I_p cos\theta, \theta \epsilon [0,2\pi], k_d \epsilon [0,1] \quad (3\text{-}2)$$

I_p为光源发出的入射光强，k_d为材质的漫反射率，θ为入射光与物体表面法向量之间的夹角，称为入射角。

镜面反射光：镜面光成分模拟的是物体表面光滑时反射的高亮的光，镜面光反映的通常是光的颜色，而不是物体的颜色。物体上一点 P 的镜面反射光的光强I_s可以表示为：

$$I_s = k_s I_p cos^n \alpha, 0 \le \alpha \le 2\pi, k_s \epsilon [0,1] \quad (3\text{-}3)$$

I_p为入射光光强，k_s为材质的镜面反射率，镜面反射光光强与$cos^n\alpha$成正比。

计算漫反射和镜面光成分时，要考虑光源和顶点位置之间向量 L、法向量 N、反射方向 R、观察者和顶点位置之间的向量 V 之间的关系。已经光线的衰弱情况。因此，Phong 光照模型可以被定义为：

$$I = k_a I_a + f(d)[k_d I_p \max(N \cdot L, 0) + k_s I_p \max(R \cdot V, 0)^n] \quad (3\text{-}4)$$

f(d)表示光照的衰弱，可以定义为：

$$f(d) = \min(1, \frac{1}{c_0 + c_1 d + c_2 d^2}) \quad (3\text{-}5)$$

c_0为常数衰减因子，c_1为线性衰减因子，c_2二次衰减因子，d 为光源位置到物体上点 P 的距离。

基于 Phong 光照模型的材质。通过多次实验，得到数据如表 3-1 所示。

表 3-1 Phong 光照模型的材质

材质名称	环境光参数（rgba）	漫发射参数（rgba）	镜面反射参数（rgba）	高光指数(float)
黄铜	R:0.329412, G:0.223529, B:0.027451, A:1.000000,	R:0.780392, G:0.568627, B:0.113725, 1.000000,	R:0.992157, G:0.941176, B:0.807843, 1.000000,	27.897400,
青铜	R:0.212500, G:0.127500, B:0.054000, A:.000000,	R:0.714000, G:0.428400, B:0.181440, 1.000000,	R:0.393548, G:0.271906, 0 B:.166721, 1.000000,	25.600000,
铬	R:0.250000, G:0.250000, B:0.250000, A:1.000000,	R:0.400000, G:0.400000, B:0.400000, A:1.000000,	R:0.774597, G:0.774597, B:0.774597, A:1.000000,	76.800003,
金	R:0.247250, G:0.199500, 0.074500, A:1.000000,	R:0.751640, G:0.606480, 0.226480, A:1.000000,	R:0.628281, G:0.555802, B:0.366065, A:1.000000,	51.200001,
翡翠	R:0.021500, G:0.174500, B:0.021500, A:0.550000,	R:0.075680, G:0.614240, B:0.075680, A:0.550000,	R:0.633000, G:0.727811, B:0.633000, A:0.550000,	76.800003,
黑曜石	R:0.053750, G:0.050000, B:0.066250, A:0.820000,	R:0.182750, G:0.170000, B:0.225250, A:0.820000,	R:0.332741, G:0.328634, B:0.346435, A:0.820000,	38.400002,

3.4 三维模型的空间变化

在渲染模型中，一个非常重要的功能就是对模型进行空间上的修改变化。把变换看成是一组顶点移动到一个新的位置的过程，这一组顶点描述成一个或者多个几何对象。尽管把一个顶点移动到另一个顶点位置有许多方式。但是把一组对象从一个位置移动到另一个位置，且他们之间的空间关系保持不变，极狐总是只有一种变换方式。因此，尽管可以找到多个矩阵，把渲染后的模型的某个顶点从一个位置移动到另一个位置，单是只存在一个矩阵，把它应用到该模型的全部的顶点后，该模型的大小仍然保持不变。

平移操作。定义平移为这样的一种操作运算，它把所有的点沿着同一个方向移动相同的距离。因此确定平移变换只需要确定一个位置矢量。平移便就是在原始向量的基础上加上另一个向量从而获得一个在不同位置的新向量的过程，从而在位移向量基础上移动了原始向量。具体的公式为：

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] * \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}6)$$

缩放变换。缩放是非刚体仿射变换，可以合适的选择缩放，平移，旋转的组合次序，得到任何仿射变换。定义缩放变换是使得几何对象变大或者变小。缩放分为两种情况。一个是各个方向上的均匀缩放变换。另一个是单个方向上的缩放变换。

缩放变换有一个固定点，因此要定义一个缩放变化，必须先确定一个固定点、缩放方向、缩放因子α。如果$\alpha > 1$，对象沿着缩放方向变大；如果α在$(0,1)$之间，对象沿着缩放方向变小；如果为α负，则表示以固定点为中心沿缩放方向的反射变换。由于确定一个固定的位置和 3 个独立缩放因子，因此缩放有 6 个自由度，具体的公式为：

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] * \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}7)$$

旋转变换。旋转是一个比较难定义的概念，它需要较多的参数。它是于标架无关且具有普遍意义。需要 3 个参数。固定点，旋转角，以及旋转轴所在的直线或者矢量。给定某个固定点，则旋转有 3 个自由度。使用三角函数，给定一个角度，可以将一个向量变换为一个经过旋转的新向量。这通常是使用一系列正弦和余弦函数（一般简称 sin 和 cos）各种巧妙的组合得到的。旋转矩阵在 3D 空间中每个单位轴都有不同定义，旋转角度用 θ 表示，则具体公式为：

绕 X 轴旋转：

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}8)$$

绕 Y 轴旋转：

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] * \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}9)$$

绕 Z 轴旋转：

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] * \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}10)$$

3.5 三维模型投影系统结果

本文的三维模型的投影系统采用 OpenGL 和 OpenCV 为几何造型平台，采用 ImGui 为 UI 界面，使用 C++语言编写。实现了一个简易渲染器，主要的功能有：读取和现实模型；旋转，缩放，移动模型；对模型添加不同的光照效果(平行光，点光源，聚光灯)；不同的材质效果；三维模型的投影。该渲染器的框架界面如图 3-8 所示。

图 3-8 渲染器的框架界面图

用户步骤包括如下几步：

步骤 1.选择需要渲染的三维模型。

步骤 2.使用鼠标来旋转，缩放，移动三维模型。

步骤 3.调节光照效果，可选择平行光，点光源，聚光灯，可以分别调

整这些光的环境光，漫反射光和镜面反射光。

步骤 4.调节三维模型的材质。

步骤 5.按下键盘’a’和’s’ 以固定水平面 60 度来旋转三维模型。

步骤 6.按下保存按钮，保存该模型的一张二维视图。

步骤 7.按下自动截图按钮，批量处理三维模型，以获取不同三维模型的二维视图集。

本文采用了 ModelNet40 作为模型数据库。对该数据库的部分模型进行渲染，渲染后的结果如图 3-9 所示。

图 3-9 部分模型

3.6 本章小结

本章主要介绍了检索系统中用到的数据集 ModelNet-40，说明了如何解析 OFF 模型文件格式，并渲染并且加入基于 Phong 光照模型的平行光，点光源和聚光灯以及材质效果。提出了基于草图检索算法的视图集选择方法。在草图检索算法中采用固定投影的方法，将获得的 6 张投影作为一个模型的二维视图集。

第4章 草图的绘制处理

设计一个简易的二维绘图画板，为不同的用户解决了在线绘制草图的功能的需求。程序界面的大小为了避免不同操作系统以及不同的分辨率，默认为 800 X 600。屏幕显示的是一张黑色的背景，可以在上面绘图，点击鼠标右键弹出绘图板的可选菜单栏，上面有不同的绘图功能以及操作功能。基本图形的绘制功能(包括了点，直线，矩形，三角形，圆形，曲线，铅笔线)；实现橡皮筋技术；基本图形的修改功能(包括了橡皮擦技术，草图保存技术)。

4.1 直线类图形绘制

Bresenham 算法用于绘制直线图。Bresenham 算法在主位移方向上每一次抖递增一个单位。另一个方向上的增量为 0 或者 1，这取决于像素点与理想直线之间的距离，这一距离定义为 d。

如图 4-1 Bresenham 直线算法所示，该直线斜率在区间 0-1 中，因此 X 方向为主要的位移方向。假设$P_i(x_i, y_i)$为当前像素，$Q(x_i+1, d)$为理想直线与下一垂直网格的交点。并且假设该直线的起点为P_i且位于网格点上，所以可以定义d_i的初始值为 0。

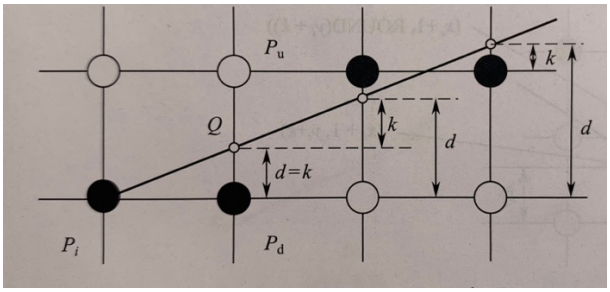

图 4-1 Bresenham 直线算法

沿着 X 方向递增一个单位，即可以得到$x_{i+1}=x_i+1$。下一个候选点为$P_d(x_i+1, y_i)$或者$P_u(x_i+1, y_i+1)$。根据 Q 点的位置来选择P_d或者P_u。而 Q 点的位置是由直线的斜率所决定的。Q 点与像素点P_d的误差项为$d_{i+1}=k$。当$d_{i+1}<0.5$时，像素P_d距离 Q 点更近，选择P_d，反之选择P_u。若 Q 点到这两个像素点的距离相等，选择任一像素均可。约定选择P_u。

因此，可以得到一个简单的的递推公式：

$$y_{i+1}=\begin{cases} y_i+1 & ,d_{i+1}\geq 0.5 \\ y_i & ,d_{i+1}<0.5 \end{cases} \quad (4\text{-}1)$$

其中，该递推公式的关键在于计算误差项d_i。沿着 X 方向递推一个单

位，有$d_{i+1}=d_i+k$。一旦 Y 方向向上走了一步，就将其减 1。由于只需要检查误差项的符号。因此，定义$e_{i+1}=d_{i+1}-0.5$，来消除小数所带来的影响。改写上述的递推公式可以得到：

$$y_{i+1}=\begin{cases}y_i+1 & ,e_{i+1}\geq 0\\ y_i & ,e_{i+1}<0\end{cases} \tag{4-2}$$

取$e_0=-0.5$。沿着 X 方向每更新一个单位，则有$e_{i+1}=e_i+k$。当e_{i+1}不小于 0 的时候，下一个像素点更新为$P_u(x_i+1,y_i+1)$。同时将e_{i+1}更新为e_{i+1}-1。反之，下一个像素点更新为$P_d(x_i+1,y_i)$。

4.1.1 直线

点击直线功能的按钮后，在绘图面板上，实现画直线的功能，在屏幕的鼠标的当前点的位置画下一个黑点，移动鼠标，按下的第一个点与此时鼠标的位置上的点，连成一条直线。当鼠标按键抬起的时候，直线生成，不再变化。当鼠标移动的时候，直线生成，但根据当前鼠标的位置进行改变。达到一个橡皮筋的效果。

4.1.2 矩形

点击矩形功能的按钮后，在绘图面板上，实现画矩形的功能，在屏幕的鼠标的当前点的位置画下一个黑点，移动鼠标，按下的第一个点与此时鼠标的位置上的点，连成一个矩形。当鼠标按键抬起的时候，矩形生成，不再变化。当鼠标移动的时候，矩形生成，但根据当前鼠标的位置进行改变。达到一个橡皮筋的效果。鼠标的两个点的位置关系如图 4-2 所示。

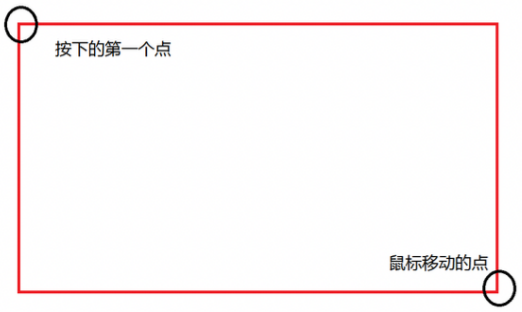

图 4-2 矩形绘制的位置关系

4.1.3 三角形

单击三角形功能的按钮后，在绘图面板上，实现绘制三角形的功能，在屏幕上的鼠标当前点绘制一个黑点，移动鼠标，第一个按下的点与此时

鼠标位置上的点相连，形成一个三角形。当鼠标按键抬起的时候，矩形生成，不再变化。当鼠标移动的时候，三角形生成，但根据当前鼠标的位置进行改变。达到一个橡皮筋的效果。鼠标的两个点的位置关系如图 4-3 所示。

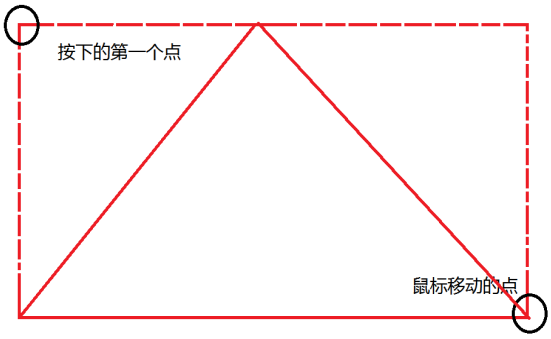

图 4-3 三角形绘制的位置关系

4.2 曲线图形绘制

选择曲线按钮可以在屏幕中通过鼠标点击任意四个点来绘制出一段三次被贝塞尔曲线。

给定 n+1 个控制点$P_i, i = 0,1,2,\cdots,n$，则 n 次贝塞尔曲线定位为：

$$p(t) = \sum_{i=0}^{n} P_i B_{i,n}(t) \quad t \in [0,1] \tag{4-3}$$

其中，$B_{i,n}(t)$为贝塞尔曲线的基函数，其表达式可以定义为：

$$B_{i,n}(t) = \frac{n!}{i!\,(n-i)!} t^i (1-t)^{n-i} = C_n^i t^i (1-t)^{n-i}, i = 0,1,2,\cdots,n \tag{4-4}$$

当 n=3 的时候，贝塞尔曲线的控制多边形有 4 个控制点，将设其分别为$P_0P_1P_2P_3$，贝塞尔曲线是三次多项式如图 4-4 所示。

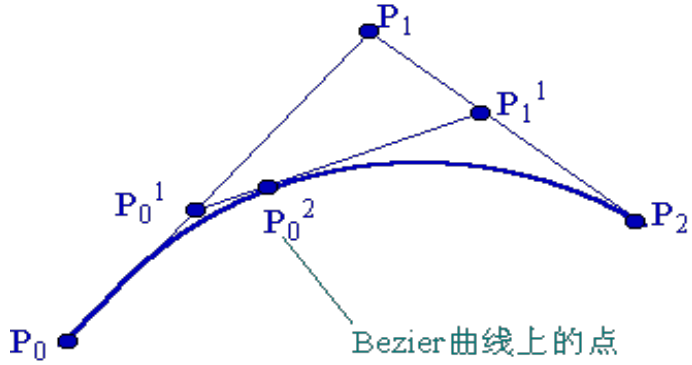

图 4-4 三次贝塞尔曲线

将该类贝塞尔曲线称其为三次贝塞尔曲线，将其定义为：

$$p(t)=\sum_{i=0}^{3}P_iB_{i,3}(t) \tag{4-5}$$

拆分开，既可以得到：

$$(1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3 \tag{4-6}$$

写成矩阵形式为：

$$p(t)=[t^3\ t^2\ t\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}P_0\\P_1\\P_2\\P_3\end{bmatrix} \tag{4-7}$$

4.3 圆形绘制

对于圆形的绘制，常用的有三种方法：第一种是采用 Bresenham 中点画圆法来绘制一个圆，具体思路与 Bresenham 绘制直线相似。第二种方法是采用细分的方法，将一个正多边形进行大量的细分，当多边形的变数足够多，多边形的边长足够小的时候，便可以得到一个近似的圆。第三种方法是采用贝塞尔曲线绘制一段圆弧，将多段圆弧拼接起来，便可以得到一个完整的圆。

本系统采用的是第三种方法，使用一段三次贝塞尔曲线可以模拟出 1/4 的单位圆。如图 4-5 所示。假定P_0^0的坐标为(0,1)，P_1^0的坐标为(m,1)，P_2^0的坐标为(1,m)，P_3^0的坐标为(1,0)。

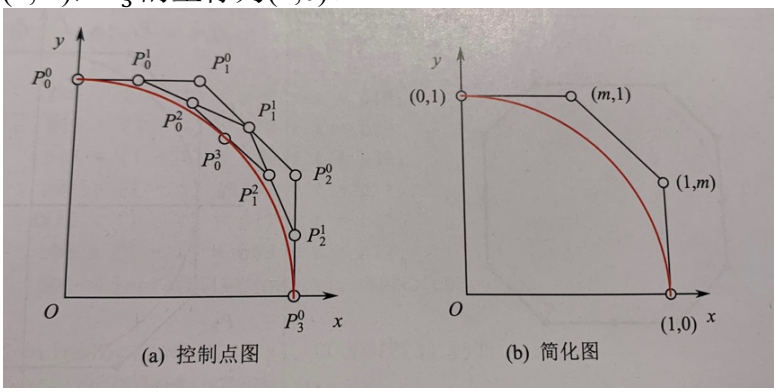

图 4-5 贝塞尔曲线模拟 1/4 圆弧

对于一段三次贝塞尔钱，其参数表达式为：

$$p(t)=(1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3 \tag{4-8}$$

将P_0^0 P_1^0 P_2^0 P_3^0代入，对于圆弧的中点，取 t=0.5，则有：

$$p\left(\frac{1}{2}\right)=\frac{1}{8}P_0+\frac{3}{8}P_1+\frac{3}{8}P_2+\frac{1}{8}P_3=\sqrt{2}/2 \quad (4\text{-}9)$$

将控制点的坐标带入，可以得到一个 m 的近似值 0.5523，称其为魔术常数。

点击圆圈功能的按钮后，在绘图面板上，实现绘制圆圈/椭圆的功能，在屏幕上鼠标的当前点绘制一个黑点，移动鼠标，第一个按下的点与此时鼠标位置上的点相连，形成一个圆。当鼠标按键抬起的时候，圆形生成，不再变化。当鼠标移动的时候，圆形生成，但根据当前鼠标的位置进行改变。达到一个橡皮筋的效果。根据鼠标的位置，显示为圆形或椭圆。鼠标的两个点的位置关系如图 4-6 所示。

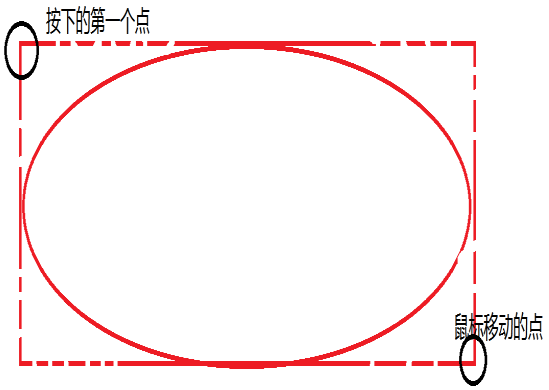

图 4-6 圆形绘制的位置关系

4.4 铅笔线绘制

点击铅笔功能的按钮后，在绘图面板上，实现现实生活中的铅笔功能，在屏幕的鼠标的当前点的位置画下一个黑点，移动鼠标，这些黑点就会连接成任意的线，可以是直线，也可以是曲线。

4.5 草图绘制的结果

本文的三维模型的投影系统采用 OpenGL 和 OpenCV 为几何造型平台，采用 MFC 为 UI 界面，使用 C++语言编写。实现了一个简易草图画板，主要的功能有：基本图形的绘制(点，直线，矩形，三角形，圆形，曲线，铅

笔线)和基本图形的修改(橡皮擦技术，草图保存技术)该简易草图画板的框架界面如图 4-7 所示。用户步骤包括如下几步：

步骤 1.点击绘制草图按钮。

步骤 2.调节弹出窗口的大小，准备绘制。

步骤 3.点击鼠标右键，弹出绘制菜单，并选择要绘制的图形。

步骤 4.绘制图形

步骤 5.可选步骤，进入上述菜单，选择修改图形或清除画板，重新绘制。

步骤 6.按下保存按钮，保存所绘制的草图。

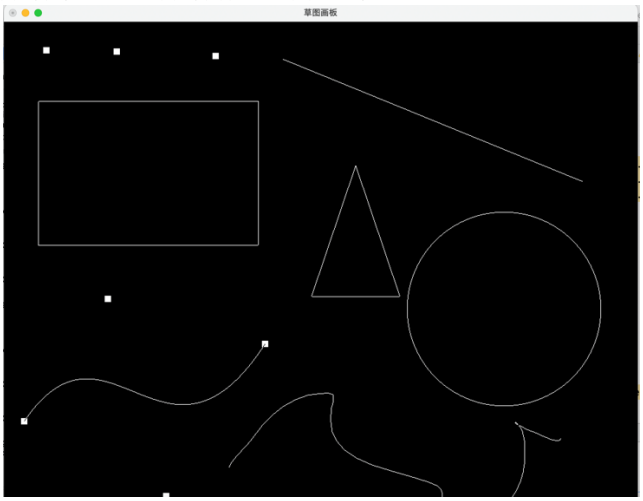

图 4-7 简易草图画板的框架

4.6 本章小结

本章主要介绍了检索系统中用到简易草图画板的设计，在该草图绘制板中加入了部分基础画图的功能，包括了直线，矩形，三角形，圆形，曲线，铅笔线，对于 Bresenham 算法和三次贝塞尔去曲线算法进行了推导。同时对这两个算法进行了应用，比如使用 Bresenham 算法来绘制直线类的图形如矩形和三角形，也对贝塞尔曲线算法进行了应用如使用多段贝塞尔曲线来绘制出一个圆形。

第5章 草图及三维模型的特征提取

在基于手绘草图的三维模型检索问题中，需要计算所有源模型与目标模型之间的相似度，以便找到与目标模型最相似的源模型。随着三维模型的数量越来越多，这样的做法显然是非常低效的。在实践过程中，我们发现直接将草图与三维模型或者三维模型的二维视图集合进行比较是没有意义的。所以需要一种能够表达二维视图和草图特征的描述符。为了能够解决二维视图和草图之间存在旋转和尺度大小的不同，本文需要的特征描述符应当具有平移、尺度和旋转不变等特性。在计算机图形学领域应用过大量的描述符，包括 HOG、SIFT、Zernike 和形状上下文等描述符。本章主要研究 Zernike 描述符、傅立叶描述符、二维形状分布和集成描述符，本章中的集成描述符可以解决单个描述符不完全特征提取的问题。

5.1 全局试图特征描述符

全局特征是指图像的整体属性，常见的全局特征包括颜色特征、纹理特征和形状特征，比如强度直方图等。由于是像素级的低层可视特征，因此，全局特征具有良好的不变性、计算简单、表示直观等特点，但特征维数高、计算量大是其致命弱点。此外，全局特征描述不适用于图像混叠和有遮挡的情况。

本文提出的全局视图特征包含 Zernike 矩和 Fourier 描述符。首先，本文提取二维视图的 Zernike 矩，并采用标准矩的方法归一化到(0, 1)范围。然后，取得二维视图的轮廓视图，提取它的一维傅立叶算子，除以直流分量标准化到(0, 1) 范围。基于草图的三维模型检索不应该受到草图绘制位置、尺度大小以及旋转角度的影响。

5.1.1 Zernike 矩

图像的矩通常描述了改图像的全局特征，并提供了大量的关于该图像不同类型的几何特征信息，比如大小，位置，方向以及形状。

在基于手绘草图的三维模型检索汇总，一个核心问题就是图像的特征提取，简单描述即为用一组较为简单的数据来描述整个图像，这组数据越简单并且越具有代表性越好。一个良好的特征矩不守光线，噪点，几何形变的干扰。

Zernike 矩是一个正交矩，是基于 Zernike 多项式正交化的函数。Zern

ike 矩具有以下几个特点：完备性，正交性，旋转不变形。Zernike 矩是一个复数矩，一般把 Zernike 矩的模作为特征来描述物体的形状。一个目标对象的特征矩可以用一组很小的 Zernike 矩特征向量来表示。低阶矩阵、特征向量描述的是一幅图像的目标的整体形状，高阶矩特征向量描述的是图像目标的细节信息。

一组定义在单位圆上的复值函数集具有完备性和正交性，使得它可以表示定义在单位圆内的任何平方可积函数，定义为：

$$V_{pq}(x,y) = V_{pq}(\rho,\theta) = R_{pq}(\rho)e^{jq\theta} \tag{5-1}$$

其中，ρ表示圆点到(x,y)的矢量长度，θ表示矢量ρ与 X 轴逆时针方向的夹角。$R_{pq}(\rho)$是实值径向多项式：

$$R_{pq}(\rho) = \sum_{s=0}^{(p-|q|)/2} (-1)^s \frac{(p-s)!}{s!\left(\frac{p+|q|}{2}-s\right)!\left(\frac{p-|q|}{2}-s\right)!}\rho^{p-2s} \tag{5-2}$$

称其为为 Zernike 多项式。

Zernike 多项式满足正交性：

$$\iint_{x^2+y^2\le 1} V_{pq}^*(x,y)V_{pq}(x,y)dxdy = \frac{\pi}{p+1}\delta_{pn}\delta_{qm} \tag{5-3}$$

其中，$V_{pq}^*(x,y)$是$V_{pq}(x,y)$的共轭多项式。并且由于 Zernike 多项式具有正交完备性，所以在单位圆内的任何图像 f(x,y)可以用唯一的式子来展开：

$$f(x,y) = \sum_{p=0}^{\infty}\sum_{q=0}^{\infty} Z_{pq}V_{pq}(\rho,\theta) \tag{5-4}$$

式子中Z_{pq}就是 Zernike 矩:

$$Z_{pq} = \frac{p+1}{\pi}\sum_x\sum_y f(x,y)\,V_{pq}(\rho,\theta) \tag{5-5}$$

针对 Zernike 矩进行平移和尺度变换得到具有平移，尺度和旋转不变性的图像 g(x,y)。g(x,y)的 Zernike 矩定义为：

$$\begin{aligned} Z_{pq} &= \frac{p+1}{\pi}\sum_x\sum_y g(x,y)\,V_{pq}(\rho,\theta) \\ &= \frac{p+1}{\pi}\sum_x\sum_y g\left(\frac{x}{q_{00}}-\bar{x},\frac{y}{q_{00}}-\bar{y}\right)V_{pq}(\rho,\theta) \end{aligned} \tag{5-6}$$

其中，$\bar{x} = \frac{q_{01}}{q_{00}}, \bar{y} = \frac{q_{10}}{q_{00}}$; q_{01}表示视图轮廓内所有黑色像素点的横坐标的和，q_{10}表示视图轮廓捏所有黑色像素点的纵坐标相加的和。q_{00}表示投影视图中所有白色像素点的总和。$(\bar{x},\bar{y})$表示手绘草图的中心。

由于 Zernike 矩定义在单位圆内，故映射在单位圆外的像素点不参与

计算，这是 Zernike 矩固有的几何误差。Zernike 矩计算时间补觉长，因此需要选择一种快速的算法来减少时间。

徐旦华[14]等提出了一种 Zernike 矩的快速算法。利用 Zernike 多项式迭代性质，找出了 Zernike 正交矩之间的内在关系，这样，高阶的 Zernike 矩可由低价的 Zernike 矩求出，再在 Chan[15]等人提出的关于一维几何矩有效算法的基础上，得出了一种快速算法。与已有方法相比，该算法大大减少了求解过程中的乘法次数，降低了计算复杂度，从而提高了运算速度和效率。

5.1.2 Fourier 描述符

傅里叶描述子的作用是用来描述图像的轮廓信息，具有平移、旋转、尺度不变性特征。对于一幅图像，通过傅里叶描述子获得其图像轮廓信息，其本质就是空间、频域变换问题。通过将图像中的像素点进行傅里叶变换，得到得到图像的轮廓信息。

如果对傅立叶描述子进行低通滤波，Fourier 描述子的低频分量捕获对象的一般形状特性，高频分量捕获更精细的细节，而且 Fourier 描述子是不考虑空间位置，因此，傅立叶描述子的作用与矩非常相似：低阶项/矩给出近似的形状，添加额外的项可以细化该形状。

傅里叶形状描述符基本思想是用物体边界的傅里叶变换作为形状描述，利用区域边界的封闭性和周期性，将二维问题转化为一维问题。由边界点导出三种形状表达，分别是曲率函数、质心距离、复坐标函数。

假设一个特定形状的边界有从 0 到 n-1 的 n 个像素。沿轮廓的第 k 个像素具的位置定义为(x_k, y_k)。对于曲线上一点，可以用复数表示：

$$\begin{aligned} S(t) &= X(t) + jY(t) \\ &= \frac{a_{x0}}{2} + \sum_{k=1}^{\infty}[a_{xk}\cos(kwt) + b_{xk}\sin(kwt)] \\ &= j[\frac{a_{y0}}{2} + \sum_{k=1}^{\infty} a_{yk}\cos(kwt) + b_{yk}\sin(kwt)] \end{aligned} \quad (5\text{-}7)$$

对 S(t)进行傅立叶变换可以得到：

$$a(k) = \sum_{t=0}^{T-1} S(t)e^{-j2\prod k\frac{t}{T}} \quad (5\text{-}8)$$

其中，a(k)为傅里叶描述子，为了使其具有平移，缩放，和旋转不变性需要对其进行归一化

归一化后的模为：

$$\frac{\|a_k\|}{\|a_1\|} \quad (5\text{-}9)$$

这个描述符具有平移、尺度和旋转不变特性。其中，平移不变性和旋

转不 变性是它本身的特性，归一化和绝对值操作保证了尺度不变性。

5.2 D2 描述符

二维形状分布是在三维形状分布的基础上变化得来的。三维形状分布的主要思想是利用形状函数来表示三维模型表面上点对的距离关系、模型内部角度 关系和区域面积，构建表达形状特征的统计分布直方图。同理，二维形状分布能够构建出二维视图的统计分布直方图。二维形状主要包括 A3、D1、D2、D3 和 D4 等五种形状函数。

A3:在模型表面随机选取三个点，三点构成的三角形内角的概率分布构成 A3 形状特征。

D1:连接模型表面任意一点，与模型中心位置产生的距离概率分布构成 D1 形状特征。

D2:在模型表面选取任意两点相连，两点之间距离的概率分布构成 D2 形状特征。

D3:在模型表面选取任意三点，三点构成的三角形面积的平方根的概率分布构成 D3 形状特征。

D4:在模型表面选取任意四点，四点构成的立方体体积的概率分布构成 D4 形状特征。不同的形状分布描述符如图 5-1 所示。

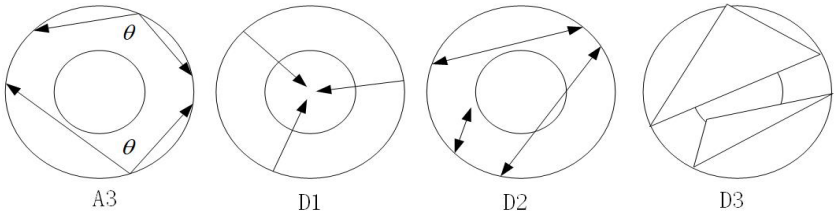

图 5-1 不同的形状分布描述符

本文采用效果较好的 D2 形状描述符来描述二维视图的形状区域特征。本文在三维模型的每个二维视图和输入草图上采用随机采样。采用的样本点$\{p_1, p_2, \dots, p_n\}$不是全部在外部的边界上，每个采样点p_i 可以位于给定视图的任意边缘线上。二维形状分布特征提取的算法描述如下:

1.在绘制草图过程中，记录草图绘制的所有像素点，把这些像素点放入集合$S = \{p(x_i, y_i) | i = 0,1,2, \dots, N-1\}$中，N 是所有像素点的总数。对像素点进行随机采样，本文选取采样个数为 1024.

2.针对 1024 个随机采样点，可以知道二维形状描述符产生所有点对的公式，具体公式如下：

$$Num = \frac{1024!}{2!\,(1024-2)!} = 523776 \qquad (5\text{-}10)$$

3.计算所有点对之间的欧几里得距离。

4. 统计随机点对之间的距离，构建二维形状分布直方图。统计分布直方图的 横坐标范围应该从 0 到点对之间的最远距离。在计算两个直方图之间的相似度时，应该保证直方图区间数目是相同的。对于区间较少的直方图，应该采用归零的方法去填充区间。

5.3 集成描述符

经过大量的实验表明傅里叶描述符中较小的值适用于表示全局视图特征。 Zernike 描述符中较大的值适用于表示全局视图特征。本文的傅里叶描述符与 Zernike 描述符均选取经验值[16]。其中，傅里叶描述符选取前 10 个较小的值; Zernike 矩描述符使用前 35 阶较大矩。对于一幅草图和三维模型的二维视图，本文使用向量的 L1 范式去度量 Fourier 描述符、Zernike 描述符和 D2 描述符之间的距离，公式为：

$$D_F = \sum_{p=1}^{10} |F_1(p) - F_2(p)|$$

$$D_Z = \sum_{q=1}^{35} |Z_1(q) - Z_2(q)| \quad (5\text{-}11)$$

$$D_D = |D_1 - D_2|$$

其中，F(p)、Z(q)分别表示草图和二维视图的傅里叶变换特征和 Zernike 矩特征。 D_1、D_2 分别表示草图和二维视图的二维分布特征。 D_F 、D_Z 和 D_D 分别为使用 Fourier 描述符、Zernike 矩描述符和 D2 形状描述符的二维草图与投影视图之间的相似性结果。其中，Fourier 描述符和 Zernike 矩描述符构成本算法的全局视图特征描述符，公式为:

$$D_G = D_F + D_Z \quad (5\text{-}12)$$

为了解决单一描述符对二维视图特征提取不全面的问题，本文在全局视图描述符和二维形状分布的基础上进行一定权重的集成，得到一个能更好地表示二维视图特征的描述符。集成描述符 D 定义为:

(5-13)

ω为全局视图描述符的权重，它与二维形状分布的权重相加为 1。草图与二维视图相似度越高，D 越小。不同用户绘画风格不同，绘画出的草图可能差异很大，集成描述子特征也会不一致，因此本文有如下方法避免检索结果的不准确。首先本文提出的集成 描述子具有平移、尺度和旋转不变等特性，避免了旋转、大小等对检索结果的 影响，因此，只有绘画

风格存在对草图的影响，而绘画风格存在两方面问题。1)绘制者对模型观察角度的不同，影响集成描述子的特征，因此本文采用多投影的方式，尽可能多的考虑不同用户观察角度的差异。2)本文草图检索对绘制有一定要求，绘制越精准检索效率越高，需要检索的准确率越高。

伊明[17]对不同的ω值进行实验。ω在[0-1]之间以每次 0.1 的速率进行增长。在不同ω值时，记录三维模型在查全率(查全率是:在检索时，检索到的所有模型占据全部模型的比例)为 0.1 时的查准率(查准率是:在检索时，检索到所求模型占据该类模型总数的比例)。ω在[0.3-0.6]范围时，集成描述子的检索性能较好;ω为 0.4 时，检索效果最佳。在后续验证本文提出算法与其它算法的优劣中，ω取值为 0.4。不同ω的检索结果如图 5-2 所示。

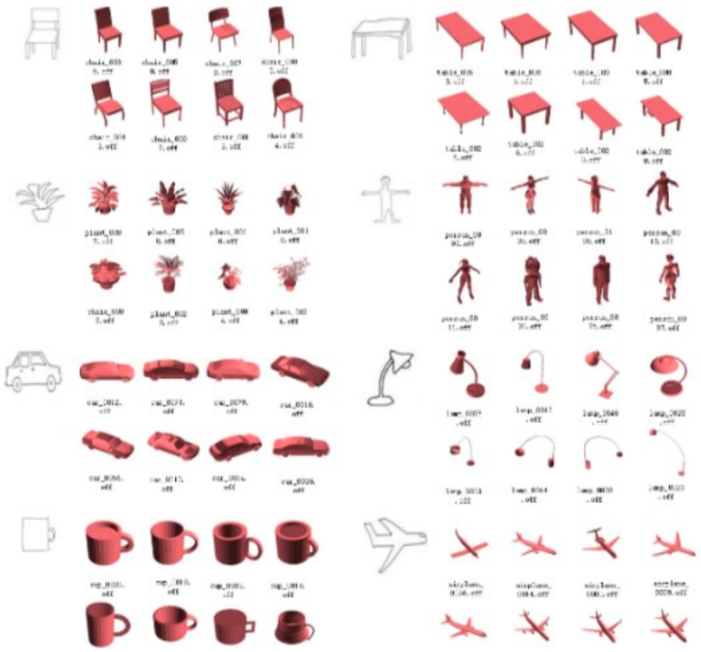

图 5-2 不同ω的检索结果

5.4 本章小结

本章主要介绍了集成描述子，并详细分析了集成描述子的特征构造，它是由 Zernike 矩和 Fourier 描述符组合的全局视图特征与 D2 形状特征按照一定比例加权得到的。集成描述子作为特征提取的描述符，能够解决单一描述符对二维草图信息描述不全的问题。

第6章 基于手绘草图的三维模型检索

相似性度量，即综合评定两个事物之间相近程度的一种度量。两个事物越接近，它们的相似性度量也就越大，而两个事物越疏远，它们的相似性度量也就越小。相似度通常表示为数值：当数据样本更相似时，相似度量越高。它通常通过转换表示为 0 和 1 之间的数字：0 表示相似性低（数据对象不同）。1 表示高度相似（数据对象非常相似）。相似性度量的给法种类繁多，一般根据实际问题进行选用。常用的相似性度是有：相关系数，角度相似性。本文中使用距离来度量样本之间的相似程度。

6.1 相似性计算

距离度量是数学上的一个基本改脸，对于任意一个定义在两个矢量 X 和 Y 上的函数 d(X,Y)只要满足如下 4 个性质就可以称作是一个距离度量。

1)非负性: d(X,Y) >= 0；

2)对称性: d(X,Y) = d(Y,X)；

3)自反性: d(X,Y) = 0，当且仅当 X= Y；

4)三角不等式: d(X,Y) + d(Y,Z) >= d(X,Z)。

常见的几种距离度量如图 6-1 所示。

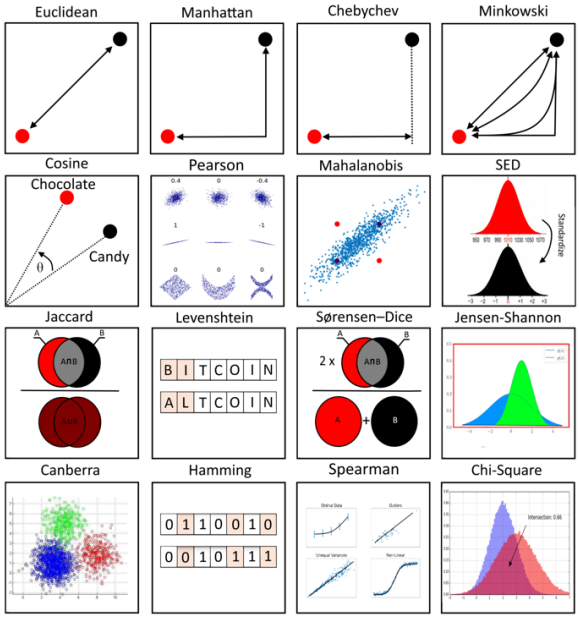

图 6-1 常见的距离度量

6.1.1 欧几里得距离

欧几里得距离也被称为欧式距离，它是一种最常见的距离度量方式:

$$d(x,y)=\left[\sum_{i=1}^{d}(x_i-y_i)^2\right]^{\frac{1}{2}} \tag{6-1}$$

欧几里得距离的直观理解是特征空间中 X 和 Y 两个点之间的直线距离，距离度量与矢量度量的长度是密切相关的。欧几里得距离也可以看作是差矢量 X-Y 的长度。矢量的长度在数学上也被称为范数，欧几里得距离对应的是矢量l_2范数，也可表示为:

$$d(x,y)=\|x-y\|_2=\sqrt{(x-y)^T(x-y)} \tag{6-2}$$

6.1.2 曼哈顿距离

曼哈顿距离又称为街区距离，也就是在欧几里德空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和。定义点 A 到点 B 的曼哈顿距离就是两点坐标之差绝对值的和。曼哈顿军力对应矢量的l_1范数，可以表示为:

$$d(x,y)=\|x-y\|_1 \tag{6-3}$$

6.1.3 切比雪夫距离

两个 N-D 观测值或向量之间的切比雪夫距离等于数据样本坐标变化的最大绝对值。在二维世界中，数据点之间的切比雪夫距离可以确定为它们的二维坐标的绝对差之和。因此，切比雪夫距离定义为：

$$d(x,y)=\max_{1\le i\le d}|x_i-y_i| \tag{6-4}$$

举例而言，在国际象棋中国王和王后所走过的两点之间最少的格数便可以用切比雪夫距离来度量，数学上切比雪夫距离对应于矢量的l_∞范数：

$$d(x,y)=\|x-y\|_\infty \tag{6-5}$$

6.1.4 闵可夫斯基距离

闵可夫斯基距离是之前距离度量的一个推广：欧几里德距离、曼哈顿距离和切比雪夫距离。它定义为 N-D 空间中两个观测值之间的距离，公式为：

$$d(x,y)=\left[\sum_{i=1}^{d}|x_i-y_i|^p\right]^{\frac{1}{p}},p\geq 1 \tag{6-6}$$

闵可夫斯基距离对应于矢量的l_p范数，不同的 p 可以得到不同的距离度量，很明显，欧几里得距离和曼哈顿距离都是闵可夫斯基距离的特例，分别对应于 p=1 和 p=2 的特殊情形，实际上，切比雪夫距离也是闵可夫斯基距离的一种特例，对应于 p 趋向于正无穷大。

在欧几里得距离度量下于坐标原点距离为 1 的点的轨迹是一个单位圆如图 6-2 所示，在曼哈顿距离度量下则变成了单位圆的内接正方形，随着闵可夫斯基距离中 p 的不断增大，单位圆不断向外扩张，直到 p 趋近于正无穷的时候，演变为外接正方形。

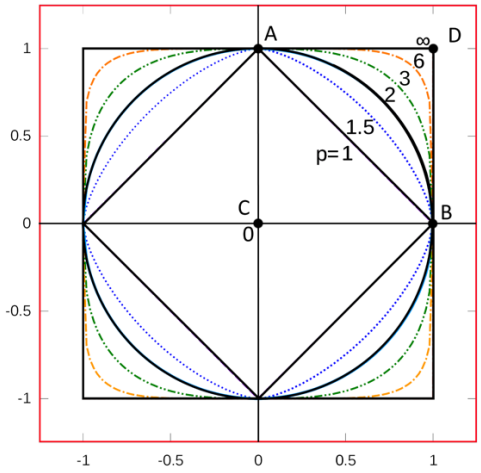

图 6-2 不同距离度量下的单位圆

6.1.5 马氏距离

上述的几种距离虽然较为简单，但也有明显的缺点，将样品的不同属性（即各指标或各变量）之间的差别等同看待，这一点有时不能满足实际要求。马氏距离表示数据的协方差距离。它是一种有效的计算两个未知样本集的相似度的方法。与欧几里得距离不同的是它考虑到各种特性之间的联系并且是尺度无关的，即独立于测量尺度。马氏距离也可以定义为两

个服从同一分布并且其协方差矩阵为 Σ 的随机变量之间的差异程度。如果协方差矩阵为单位矩阵，那么马氏距离就简化为欧氏距离，如果协方差矩阵为对角阵，则其也可称为正规化的欧氏距离。
对于一个均值为μ，协方差矩阵为 Σ 的多变量矢量 x，其马氏距离为:

$$D_M(x) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)} \quad (6\text{-}7)$$

6.2 基于手绘草图的三维模型检索结果

在分析了上述的距离计算方法后，本文采用了最为简单的欧几里得距离公式来计算草图和二维视图集的相似性。基于该相似性算法，设计了基于手绘草图的三维模型检索。

本文的三维模型的检索系统采用 OpenGL 和 OpenCV 为几何造型平台，采用 MFC 为 UI 界面，使用 C++语言编写。实现了一个检索程序系统，主要的功能有：导入和绘制草图，绘制草图的功能的具体实现参照第 4 章的简易草图画板的实现；可以选择不同的数据库；显示出绘制的草图；图形检索功能；显示 8 张最优的结果图，以及一张全局最优结果。该检索器的框架界面如图 6-3 所示。

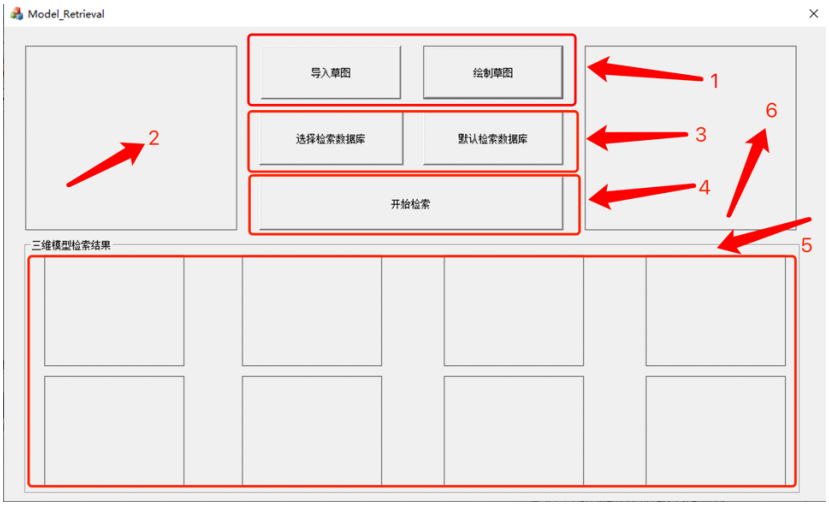

图 6-3 检索器的框架界面

用户步骤包括如下几步:

步骤 1.用户选择绘制在线绘制草图，或者导入已经绘制好的草图的图片。如果按下绘制草图，会弹出一个简易草图绘制画板供用户绘制草图，具体功能参照第 4 章。

步骤 2.在程序的左侧会显示出用户导入或者绘制好的草图。

步骤 3.选择检索三维模型库路径，默认有一个较小的数据库，用户可以通过简易渲染器的程序来获得不同数量级的模型数据库。简易渲染器的

具体功能参照第 3 章。

步骤 4.点击检索，进行草图检索。

步骤 5.输出结果，返回 8 张结果图。

步骤 6.输出结果，返回 1 张最接近的结果图。以一个显示器为例子，进行基于手绘草图的检索，检索的结果如图 6-4 所示。

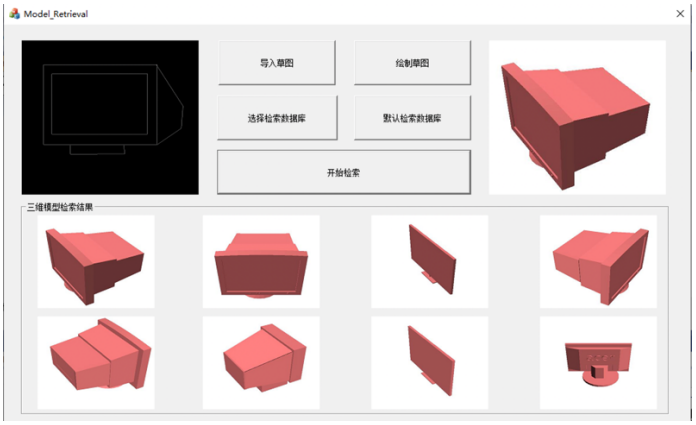

图 6-4 草图检索结果

6.3 本章小结

本章主要介绍了基于不同距离公式的相似性度量，推到了不同距离公式的计算方法和范数。本文利用欧式距离计算二维视图与草图之间的相似性，对手绘草图的检索结果进行从小到大的排序显示。本章还着重介绍了基于手绘草图的三维模型检索系统的应用。针对草图的检索过程的第一步为绘制草图，为了降低了对用户的要求，提供了在线绘制或者线下绘制的功能，只需要简单的草图即可完成检索，实验结果也表明本文的方法可以有效地检索相关的三维模型。

结论

随着互联网技术和计算机辅助设计的飞速发展，三维 CAD 模型数量的剧增，如何从海量模型数据中快速、准确的检索出所需模型变得尤为重要。对现有工业三维模型的有效检索能够促进新模型的设计，节省大量的时间和人力物力。计算机辅助设计（Computer Aided Design，CAD）利用计算机及其辅助设备帮助设计人员进行计算、信息存储和图形设计等工作，从而极大的减少了设计人员的工作量，缩短了产品设计周期并提高了产品设计质量。

本文描述了三维模型检索技术在国内外的研究现状，对三维模型检索技术的发展和主要方法作了概述。本文主要研究了基于手绘草图的三维模型检索。本文在基于草图检索的过程中，首先利用 OpenGL 来渲染出模型，并采用固定投影的方式来获取模型点 6 张二维视图集。提供给用户不同的绘制方法，来使得用户可以线上或者线下绘制草图以此来提高识别准确率。使用了一种新的描述子，即集成描述子，该描述子有效地利用了视图的区域特征和边界轮廓特征，最后利用距离公式完成检索。 实验验证本文方法效果更好。基于草图的检索算法相对于其他算法的优势在于便于普通用户检索，不需要拥有大量三维模型的细节信息就可以检索出较为准确的三维模型。

本文利用 OpenGL 和 OpenCV 为几何造型平台，采用 MFC 和 ImGui 为 UI 界面，采用 C++ 为编程语言开发了一个三维模型检索。该系统集成了本文提出的所有算法，包括了简易模型的渲染，简易草图的绘制，三维模型检索。利用 ModelNet-40 完成模型库，验证了本文提出算法的优良性能和检索系统的人性化开发。

上述研究取得了一些阶段性的成果，但也还存在许多需要进一步研究的内容，主要有:

1.由于有些三维模型的结构很复杂，不同的用户在绘制草图时候容易带来一定的误差因素，如笔画的闭合等，因此需要降噪处理，手绘草图降噪的目的是将用户绘画时不小心或者无意识造成的聚点，封闭处的缝隙，提笔或者收笔造成的毛刺消除，防止手绘草图过于粗糙，导致手绘草图离用户原本意图差异较大。

2.由于现在的三维模型的构成复杂，采用的相似性计算为欧几里得距离，因此，当模型数量急剧增大时，检索的准确率会有一定程度的下降

致谢

时光荏苒,岁月如梭。四年的大学生活马上就要结束了，有太多的收获和不舍。春梦秋云，聚散无常。随着离校日期的日趋渐近，毕业论文的完成也随之进入了尾声。在这短短四年的求学旅途中，不仅学习到了很多在未来工作中将要用到的知识技能，还让我学会了一些为人处世的经验，收获到了和同学间宝贵的友情。

感谢我非常尊敬的导师高雪瑶老师。本论文在高老师的悉心指导下完成的。高老师在学术研究上秉承着专注负责、 精益求精的治学态度，诲人不倦的高尚师德。不仅使本人树立了远大的学习目标，还使本人明白了学多为人处事的道理。在校期间，在高老师的教导下，也逐渐做了几个项目，收获良多。本次论文从宣帝到完成，都离不开导师的良苦用心，倾注了老师大量的心血。在此，谨向高老师表达由衷的感谢。感谢大学中其他老师在学习和生活当中对我的关心和帮助。感谢于林森老师、李莎莎老师在课后听我询问问题和耐心的解答，感谢所有老师提供了一个学习氛围浓厚的校园环境，

四年前的一次选择，便一人独自跨越半个中国来到哈尔滨求学。非常荣幸能够在哈尔滨理工大学完成自己本科学习。求学生涯最想感谢的是我的父母，求学以来，已经过了16年，是他们的默默付出，鼓舞着我学习。感谢我的母亲为了让我在更好的学习环境中，走遍了家乡城市的所有学校。感谢我的父亲，在我初高中阶段，每周开车送我往返于石狮和厦门两个城市之间。在哈理工的四年中，感谢我的 5 个舍友们，你们一直的陪伴让我明白了友情的可贵。

感谢答辩组中的所有老师，感谢老师们在百忙之中可以抽出时间参加硕士本科生的毕业答辩，祝老师们在今后的工作和生活中健健康康，心想事成，万事如意。

参考文献

[1] Sezgin T M, Stahovich T, Davis R. Sketch based interfaces: early pro cessing for sketch understanding[C].The Workshop on Perceptive User Inte rfaces.ACM,2001:1-8
[2] Li B, Lu Y, Fares R. Semantic sketch-based 3D model retrieval[C]. I EEE International Conference on Multimedia and Expo Workshops. IEEE, 2013:555-558
[3] Yuan Juefei, Abdul Rashid Hameed, Li Boet al.A comparison of meth ods for 3D scene shape retrieval[J] Computer Vision and Image Understan ding, 2020,201:103-120
[4] Zhu Z, Wang X, Bai S, et al. Deep learning representation using auto encoder for 3D shape retrieval[J]. Neurocomputing, 2016,204:41-50.
[5] Konstantinos Sfikas,Ioannis Pratikakis,Theoharis Theoharis.Ensemble of PANORAMA-based convolutional neural networks for 3D model classificat ion and retrieval[J]Computers & Graphics.2018:208-218
[6] 周燕,曾凡智,吴臣,罗粤,刘紫琴.基于深度学习的三维形状特征提取方法[J].计算机科学.2019(09):47-58
[7] 张云峰.图像与三维模型匹配方法的研究及应用[D].南京:南京大学,2019.
[8] 张艺琨,唐雁,陈强.基于多特征融合的三维模型检索[J].郑州大学学报(工学版).2019(01):1-6
[9] 李海生,孙莉,吴晓群,蔡强,杜军平.基于模型内二面角分布直方图的非刚性三维模型检索[J].计算机辅助设计与图形学学报.2017(06): 1128-1134
[10] 安勃卿.基于手绘草图的三维模型检索研究与实现[D].西北大学,2017.
[11] Su H，Maji S，Kalogerakis Eet al.Multi-view convolutional neural n etworks for 3d shape recognition[C]//Pro- ceedingsof ICCV2015，2015
[12] C. M. Cyr and B. B. Kimia. A similarity-based aspect-graph approac h to 3D object recognition. 57(1), 2004. 3
[13] 潘婷.基于草图的三维模型检索方法研究与应用[D].中北大学 2020
[14] 徐旦华,辜嘉,李松毅,舒华忠.Zernike 矩的快速算法[J]东南大学学报 200 2(03):32:2
[15] Chan FHY，Lam FK.An all adder systolic structure for fast computa tion of moments[J].JVLSI Signal Process，196，12:159 175.
[16] DALAL N, TRIGGS B. Histograms of Oriented Gradients for Huma n Detection[C]. Computer Vision and Pattern Recognition, 2005. CVPR 20 05. IEEE Computer Society Conference on, 2005, 1: 886-893.
[17] 伊明.基于内容与草图的三维模型检索方法[D].哈尔滨理工大学.2020
[18] 强会英, 李雨虹, 王洪申, 等. 基于 HMM 和仿射不变矩三维模型归类与检 索算法[J]. 制造业自动化, 2019, 41(3): 40-46.

附录 A

附录 A 主要为三维模型投影程序的部分主要代码的实现

1.模型的光照 CLight.h

```
#pragma once
#include "ggl.h"
class CLight
{
public:
    CLight();
    ~CLight();
public:
    GLenum m_LightIdentifier;
public:
    void SetAmbientColor(float r, float g, float b,
float a);
    void SetDiffuseColor(float r, float g, float b,
float a);
    void SetSpecularColor(float r, float g, float b,
float a);
    void Enable();
    void Disable();
};
class CDirectionLight : public CLight
{
public:
    CDirectionLight(GLenum light);
    void SetPosition(float x, float y, float z);
};
class CPointLight : public CLight
{
public:
    CPointLight(GLenum light);
    void SetPosition(float x, float y, float z);
    void SetConstAttenuation(float v);
    void SetLinearAttenuation(float v);
    void SetQuadricAttenuation(float v);
};
class CSpotLight : public CPointLight
{
public:
```

```
    CSpotLight(GLenum light);
    void SetDirection(float x, float y, float z);
    void SetExponent(float v);
    void SetCutoff(float v);
};
```

2.模型的光照 CLight.cpp

```
#include "CLight.h"
void CLight::SetAmbientColor(float r, float g, float b,
float a)
{
    float ambientColor[] = { r, g, b, a };
    glLightfv(m_LightIdentifier,            GL_AMBIENT,
ambientColor);
}
void CLight::SetDiffuseColor(float r, float g, float b,
float a)
{
    float diffuseColor[] = { r, g, b, a };
    glLightfv(m_LightIdentifier,            GL_DIFFUSE,
diffuseColor);
}
void CLight::SetSpecularColor(float r, float g, float
b, float a)
{
    float sepcularColor[] = { r, g, b, a };
    glLightfv(m_LightIdentifier,           GL_SPECULAR,
sepcularColor);
}
void CLight::Enable()
{
    glEnable(GL_LIGHTING);
    glEnable(m_LightIdentifier);
}
void CLight::Disable()
{
    glDisable(GL_LIGHTING);
    glDisable(m_LightIdentifier);
}
CDirectionLight::CDirectionLight(GLenum light)
{
    m_LightIdentifier = light;
}
```

```
void CDirectionLight::SetPosition(float x, float y,
float z)
{
    float pos[] = { x, y, z, 0.0f };
    glLightfv(m_LightIdentifier, GL_POSITION, pos);
}
CPointLight::CPointLight(GLenum light)
{
    m_LightIdentifier = light;
}
void CPointLight::SetPosition(float x, float y, float
z)
{
    float pos[] = { x, y, z, 1.0f };
    glLightfv(m_LightIdentifier, GL_POSITION, pos);
}
void CPointLight::SetConstAttenuation(float v)
{
    glLightf(m_LightIdentifier,
GL_CONSTANT_ATTENUATION, v);
}
void CPointLight::SetLinearAttenuation(float v)
{
    glLightf(m_LightIdentifier, GL_LINEAR_ATTENUATION,
v);
}
void CPointLight::SetQuadricAttenuation(float v)
{
    glLightf(m_LightIdentifier,
GL_QUADRATIC_ATTENUATION, v);
}
CSpotLight::CSpotLight(GLenum          light)          :
CPointLight(light)
{}
void CSpotLight::SetDirection(float x, float y, float
z)
{
    float dir[] = { x, y, z, 1.0f };
    glLightfv(m_LightIdentifier,     GL_SPOT_DIRECTION,
dir);
}
void CSpotLight::SetExponent(float v)
{
```

```
    glLightf(m_LightIdentifier, GL_SPOT_EXPONENT, v);
}
void CSpotLight::SetCutoff(float v)
{
    glLightf(m_LightIdentifier, GL_SPOT_CUTOFF, v);
}
```

3. 模型的加载 CMyModle.h

```
#pragma once
#include "ggl.h"
typedef struct Vertex
{
    float x;
    float y;
    float z;
} Vertex;
typedef struct Face
{
    Face(void) : vert_number(0), verts(0) {};
    int vert_number;
    Vertex** verts;
    float normal[3];
} Face;

typedef struct MyMesh
{
    MyMesh(void) : vert_number(0), verts(0),
face_number(0), faces(0) {};
    int vert_number;
    Vertex* verts;
    int face_number;
    Face* faces;
    vector<Vertex> point;
} MyMesh;
class CMyModel
{
public:
    CMyModel();
    CMyModel(const char* filename);
    ~CMyModel();
public:
    MyMesh* ReadOffFile();
    MyMesh* ReRead(const char* newpath);
```

```
    void get_normal(Face& face);
    void draw_faces();
    void draw_points();
    void draw_lines();
    void SetAmbientMaterial(float r, float g, float b,
float a);
    void SetDiffuseMaterial(float r, float g, float b,
float a);
    void SetSpecularMaterial(float r, float g, float
b, float a);
    void SetShininessMaterial(float s);
    void SetMaterial();
public:
    const char* m_filename;
    MyMesh* m_mesh;
    float m_AmbientMaterial[4];
    float m_DiffuseMaterial[4];
    float m_SpecularMaterial[4];
    float m_ShininessMaterial[1];
};

4.CMyModel.cpp
#include "CMyModel.h"
CMyModel::CMyModel(const char* filename)
{
    m_filename = filename;
    m_mesh = NULL;
    memset(m_AmbientMaterial, 0,
sizeof(m_AmbientMaterial));
    memset(m_DiffuseMaterial, 0,
sizeof(m_DiffuseMaterial));
    memset(m_SpecularMaterial, 0,
sizeof(m_SpecularMaterial));
    memset(m_ShininessMaterial, 0,
sizeof(m_ShininessMaterial));
}
void CMyModel::get_normal(Face& face)
{
    face.normal[0] = face.normal[1] = face.normal[2] =
0;
    Vertex* v1 = face.verts[face.vert_number - 1];
    for (int i = 0; i < face.vert_number; i++)
    {
```

```
        Vertex* v2 = face.verts[i];
        face.normal[0] += (v1->y * v2->z) - (v1->z *
v2->y);
        face.normal[1] += (v1->z * v2->x) - (v1->x *
v2->z);
        face.normal[2] += (v1->x * v2->y) - (v1->y *
v2->x);
        v1 = v2;
    }
    float squared_normal_length = 0.0;
    squared_normal_length += face.normal[0] *
face.normal[0];
    squared_normal_length += face.normal[1] *
face.normal[1];
    squared_normal_length += face.normal[2] *
face.normal[2];
    float normal_length = sqrt(squared_normal_length);
    if (normal_length > 1.0E-6)
    {
        face.normal[0] /= normal_length;
        face.normal[1] /= normal_length;
        face.normal[2] /= normal_length;
    }
}
MyMesh* CMyModel::ReadOffFile()
{
    FILE* fp = NULL;
    errno_t err;

    if (!(fp = fopen(m_filename, "r")))
    {
        cout << "无法打开文件" << endl;
        return 0;
    }
    m_mesh = new MyMesh();
    int vert_number = 0;
    int face_number = 0;
    int line_number = 0;
    int line_count = 0;
    char buffer[1024];
    while (fgets(buffer, 1023, fp))
    {
        line_count++;
```

```
        char* bufferp = buffer;
        while (isspace(*bufferp)) bufferp++;
        if (*bufferp == '#') continue;
        if (*bufferp == '\0') continue;
        if (vert_number == 0)
        {
            if (!strstr(bufferp, "OFF"))
            {
                if ((sscanf(bufferp, "%d%d%d",
&vert_number, &face_number, &line_number) != 3) ||
(vert_number == 0)) {
                    cout << "ERROR: 存在语法错误! " <<
endl;
                    fclose(fp);
                    return NULL;
                }
                m_mesh->verts = new Vertex[vert_number];
                assert(m_mesh->verts);
                m_mesh->faces = new Face[face_number];
                assert(m_mesh->faces);
            }
        }
        else if (m_mesh->vert_number < vert_number)
        {
            Vertex& vert =
m_mesh->verts[m_mesh->vert_number++];
            if (sscanf(bufferp, "%f%f%f", &(vert.x),
&(vert.y), &(vert.z)) != 3)
            {
                cout << "ERROR: 点的信息中，数据量不足（3个）
" << endl;
                fclose(fp);
                return NULL;
            }
        }
        else if (m_mesh->face_number < face_number)
        {
            char* pTmp = NULL;
            Face& face =
m_mesh->faces[m_mesh->face_number++];
            bufferp = strtok(bufferp, " \t");
            if (bufferp)
                face.vert_number = atoi(bufferp);
```

```
            else
            {
                fclose(fp);
                return NULL;
            }
            face.verts = new Vertex *
[face.vert_number];
            assert(face.verts);
            for (int i = 0; i < face.vert_number; i++)
            {
                char* pTmp = NULL;
                bufferp = strtok(NULL, " \t");
                if (bufferp)
                    face.verts[i] =
&(m_mesh->verts[atoi(bufferp)]);
                else
                {
                    fprintf(stderr, "ERROR: Syntax error
with face on line %d in file %s\n", line_count,
m_filename);
                    fclose(fp);
                    return NULL;
                }
            }
            get_normal(face);
        }
        else
        {
            cout << "ERROR: 格式存在错误！" << endl;
            break;
        }
    }
    if (face_number != m_mesh->face_number)
    {
        cout << "ERROR: 面的数目与实际不符" << endl;
    }
    fclose(fp);
    return m_mesh;
}
MyMesh* CMyModel::ReRead(const char* newpath)
{
    memset(m_mesh, 0, sizeof(m_mesh));
    m_filename = newpath;
```

```
    m_mesh = ReadOffFile();
    return m_mesh;
}
void CMyModel::draw_faces()
{
    //if (m_mesh != NULL)
    {
        for (int i = 0; i < m_mesh->face_number; i++)
        {
            Face& face = m_mesh->faces[i];
            glColor3f(1.0f, 1.0f, 1.0f);
            glBegin(GL_POLYGON);
            glNormal3fv(face.normal);
            for (int j = 0; j < face.vert_number; j++)
            {
                Vertex* vert = face.verts[j];
                glVertex3f(vert->x, vert->y, vert->z);
            }
            glEnd();
        }
    }
}
void CMyModel::draw_points()
{
    glColor3f(1.0f, 1.0f, 1.0f);
    glPointSize(3);
    glBegin(GL_POINTS);
    for (int i = 0; i < m_mesh->vert_number; i++)
    {
        glVertex3f(m_mesh->verts[i].x,
            m_mesh->verts[i].y,
            m_mesh->verts[i].z);
    }
    glEnd();
}
void CMyModel::draw_lines()
{
    double temp_x, temp_y, temp_z;
    for (int i = 0; i < m_mesh->face_number; i++)
    {
        Face& face = m_mesh->faces[i];
        glColor3f(1.0f, 1.0f, 1.0f);
        glBegin(GL_LINES);
```

```
        for (int j = 0; j < face.vert_number; j++)
        {
            Vertex* vert = face.verts[j];
            if (j == 0) {
                temp_x = vert->x;
                temp_y = vert->y;
                temp_z = vert->z;
                continue;
            }
            glVertex3f(temp_x, temp_y, temp_z);
            glVertex3f(vert->x, vert->y, vert->z);
            temp_x = vert->x;
            temp_y = vert->y;
            temp_z = vert->z;
        }
        glEnd();
    }
}
void CMyModel::SetAmbientMaterial(float r, float g,
float b, float a)
{
    m_AmbientMaterial[0] = r;
    m_AmbientMaterial[1] = g;
    m_AmbientMaterial[2] = b;
    m_AmbientMaterial[3] = a;
}
void CMyModel::SetDiffuseMaterial(float r, float g,
float b, float a)
{
    m_DiffuseMaterial[0] = r;
    m_DiffuseMaterial[1] = g;
    m_DiffuseMaterial[2] = b;
    m_DiffuseMaterial[3] = a;
}
void CMyModel::SetSpecularMaterial(float r, float g,
float b, float a)
{
    m_SpecularMaterial[0] = r;
    m_SpecularMaterial[1] = g;
    m_SpecularMaterial[2] = b;
    m_SpecularMaterial[3] = a;
}
```

```
void CMyModel::SetShininessMaterial(float s)
{
    m_ShininessMaterial[0] = s;
}
void CMyModel::SetMaterial()
{
    glMaterialfv(GL_FRONT, GL_AMBIENT,
m_AmbientMaterial);
    glMaterialfv(GL_FRONT, GL_DIFFUSE,
m_DiffuseMaterial);
    glMaterialfv(GL_FRONT, GL_SPECULAR,
m_SpecularMaterial);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS,
m_ShininessMaterial);
}
```

附录 B

附录 B 主要为简易绘制绘图板程序的部分主要代码的实现

1. CMyDraw.h

```
#pragma once
#include "ggl.h"
typedef struct MYPOS
{
    GLfloat x;
    GLfloat y;
}MYPOS;
class CMyDraw
{
public:
    CMyDraw()    {}
    ~CMyDraw()    {};
public:
    void Draw_MouseDown(int hh, int x, int y)
    {
        m_MouseBegin.x = x;
        m_MouseBegin.y = hh - y;
        m_MouseEnd.x = m_MouseBegin.x;
        m_MouseEnd.y = m_MouseBegin.y;
    }
    virtual void Draw_MouseUp(int hh, int x, int y) =
0;
    virtual void Draw_MouseMove(int hh, int x, int y)
= 0;
public:
    MYPOS m_MouseBegin;
    MYPOS m_MouseEnd;
    MYPOS m_MouseMove;
    int hh;
};

class CLine : public CMyDraw
{
public:
    CLine()    {}
    ~CLine()    {}
public:
```

```
    void Draw_MouseUp(int hh, int x, int y);
    void Draw_MouseMove(int hh, int x, int y);
};
class CPoint2 : public CLine
{
public:
    CPoint2()    {}
    ~CPoint2()    {}
public:
    void Draw_MouseDown(int hh, int x, int y);
    void Draw_MouseUp(int hh, int x, int y);
public:
    float x, y;
    void setxy(float x2, float y2)
    {
        x = x2;
        y = y2;
    }
    CPoint2 operator&(const CPoint2& rPoint)
    {
        x = rPoint.x;
        y = rPoint.y;
        return *this;
    }
};
class CPencil : public CLine
{
public:
    CPencil()    {}
    ~CPencil()    {}
public:
    void Draw_MouseMove(int hh, int x, int y);
};
class CCircle : public CMyDraw
{
public:
    CCircle()
    {}
    ~CCircle()
    {}
public:
    void Draw_MouseUp(int hh, int x, int y);
    void Draw_MouseMove(int hh, int x, int y);
```

```
public:
    float m_radius;
};
class CRectangle : public CMyDraw
{
public:
    CRectangle()    {}
    ~CRectangle()    {}
public:
    void Draw_MouseUp(int hh, int x, int y);
    void Draw_MouseMove(int hh, int x, int y);
};
class CTriangle : public CMyDraw
{
public:
    CTriangle()    {}
    ~CTriangle()    {}
public:
    void Draw_MouseUp(int hh, int x, int y);
    void Draw_MouseMove(int hh, int x, int y);
};
class CCurve : public CMyDraw
{
public:
    CCurve()
    {
        nNumPoints = 4;
        currentNum = 0;
    }
    ~CCurve()    {}
public:
    int nNumPoints;
    int currentNum;
    float ctrlPoint[][3];
public:
    void Draw_MouseDown(int hh, int x, int y);
    void Draw_MouseUp(int hh, int x, int y);
    void Draw_MouseMove(int hh, int x, int y);
};
```

2. CMyDraw.cpp

```
#include "CMyDraw.h"
void CLine::Draw_MouseUp(int hh, int x, int y)
```

```
{
    m_MouseEnd.x = x;
    m_MouseEnd.y = hh - y;
    glDisable(GL_COLOR_LOGIC_OP);
        glBegin(GL_LINES);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glEnd();
    glFlush();
}
void CLine::Draw_MouseMove(int hh, int x, int y)
{
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_XOR);//用异或的方式来画
     glBegin(GL_LINES);//两点确定一条直线
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glEnd();
    m_MouseMove.x = x;
    m_MouseMove.y = y;
    m_MouseEnd.x = m_MouseMove.x;
    m_MouseEnd.y = hh - m_MouseMove.y;
    glBegin(GL_LINES);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glEnd();
    glFlush();
}
void CCircle::Draw_MouseUp(int hh, int x, int y)
{
    m_MouseEnd.x = x;
    m_MouseEnd.y = hh - y;
    glDisable(GL_COLOR_LOGIC_OP);
    m_radius = sqrt((m_MouseBegin.x - m_MouseEnd.x) *
(m_MouseBegin.x - m_MouseEnd.x) + (m_MouseBegin.y -
m_MouseEnd.y) * (m_MouseBegin.y - m_MouseEnd.y) *
1.0) / 2;
    glBegin(GL_LINE_LOOP);
    for(int i=0; i < 360; i++)
    {
        glVertex2f((m_MouseBegin.x + m_MouseEnd.x) / 2
+ m_radius * GLfloat(sin(2 * PI * i / 360)),
                (m_MouseBegin.y + m_MouseEnd.y) / 2 +
```

```
m_radius * GLfloat(cos(2 * PI * i / 360)));
    }
    glEnd();
    glFlush();
}
void CCircle::Draw_MouseMove(int hh, int x, int y)
{
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_XOR);
    m_radius = sqrt((m_MouseBegin.x - m_MouseEnd.x) *
(m_MouseBegin.x - m_MouseEnd.x) + (m_MouseBegin.y -
m_MouseEnd.y) * (m_MouseBegin.y - m_MouseEnd.y) *
1.0) / 2;
    glBegin(GL_LINE_LOOP);
    for(int i=0; i < 360; i++)
    {
        glVertex2f((m_MouseBegin.x + m_MouseEnd.x) / 2
+ m_radius * GLfloat(sin(2 * PI * i / 360)),
                (m_MouseBegin.y + m_MouseEnd.y) / 2 +
m_radius * GLfloat(cos(2 * PI * i / 360)));
    }
    glEnd();
    m_MouseMove.x = x;
    m_MouseMove.y = y;
    m_MouseEnd.x = m_MouseMove.x;
    m_MouseEnd.y = hh - m_MouseMove.y;
    m_radius = sqrt((m_MouseBegin.x - m_MouseEnd.x) *
(m_MouseBegin.x - m_MouseEnd.x) + (m_MouseBegin.y -
m_MouseEnd.y) * (m_MouseBegin.y - m_MouseEnd.y) *
1.0) / 2;
    glBegin(GL_LINE_LOOP);
    for(int i=0; i < 360; i++)
    {
        glVertex2f((m_MouseBegin.x + m_MouseEnd.x) / 2
+ m_radius * GLfloat(sin(2 * PI * i / 360)),
                (m_MouseBegin.y + m_MouseEnd.y) / 2 +
m_radius * GLfloat(cos(2 * PI * i / 360)));
    }
    glEnd();
    glFlush();
}
void CRectangle::Draw_MouseUp(int hh, int x, int y)
{
```

```
    m_MouseEnd.x = x;
    m_MouseEnd.y = hh - y;
    glDisable(GL_COLOR_LOGIC_OP);
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseBegin.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseBegin.y);
    glEnd();
    glFlush();
}
void CRectangle::Draw_MouseMove(int hh, int x, int y)
{
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_XOR);
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseBegin.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseBegin.y);
    glEnd();
    m_MouseMove.x = x;
    m_MouseMove.y = y;
    m_MouseEnd.x = m_MouseMove.x;
    m_MouseEnd.y = hh - m_MouseMove.y;
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseBegin.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(m_MouseEnd.x, m_MouseBegin.y);
    glEnd();
    glFlush();
}
void CTriangle::Draw_MouseUp(int hh, int x, int y)
{
    m_MouseEnd.x = x;
    m_MouseEnd.y = hh - y;
    glDisable(GL_COLOR_LOGIC_OP);
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(abs(m_MouseBegin.x - abs(m_MouseEnd.x -
m_MouseBegin.x)), m_MouseEnd.y);
```

```
    glEnd();
    glFlush();
}
void CTriangle::Draw_MouseMove(int hh, int x, int y)
{
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_XOR);
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(abs(m_MouseBegin.x - abs(m_MouseEnd.x -
m_MouseBegin.x)), m_MouseEnd.y);
    glEnd();
    m_MouseMove.x = x;
    m_MouseMove.y = y;
    m_MouseEnd.x = m_MouseMove.x;
    m_MouseEnd.y = hh - m_MouseMove.y;
    glBegin(GL_LINE_LOOP);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glVertex2i(abs(m_MouseBegin.x - abs(m_MouseEnd.x -
m_MouseBegin.x)), m_MouseEnd.y);
    glEnd();
    glFlush();
}
void CPoint2::Draw_MouseDown(int hh, int x, int y)
{
    m_MouseBegin.x = x;
    m_MouseBegin.y = hh - y;

    glPointSize( 10 );
    glBegin(GL_POINTS);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glEnd();
    glFlush();
}
void CPoint2::Draw_MouseUp(int hh, int x, int y)
{
    glBegin(GL_POINT);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glEnd();
    glFlush();
}
```

```
void CPencil::Draw_MouseMove(int hh, int x, int y)
{
    m_MouseMove.x = x;
    m_MouseMove.y = hh - y;
    m_MouseBegin.x = m_MouseEnd.x;
    m_MouseBegin.y = m_MouseEnd.y;
    m_MouseEnd.x = m_MouseMove.x;
    m_MouseEnd.y = m_MouseMove.y;
       glPointSize( 1 );
    glBegin(GL_LINES);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glVertex2i(m_MouseEnd.x, m_MouseEnd.y);
    glEnd();
    glFlush();
}
void CCurve::Draw_MouseDown(int hh, int x, int y)
{
    m_MouseBegin.x = x;
    m_MouseBegin.y = hh - y;

    glPointSize( 10 );
    glBegin(GL_POINTS);
    glVertex2i(m_MouseBegin.x, m_MouseBegin.y);
    glEnd();
       if(currentNum < nNumPoints)
    {
       ctrlPoint[currentNum][0] = m_MouseBegin.x;
       ctrlPoint[currentNum][1] = m_MouseBegin.y;
       ctrlPoint[currentNum][2] = 0.0f;
       currentNum++;
    }
    if(currentNum == 4)
    {
       currentNum = 0;
       glMap1f(GL_MAP1_VERTEX_3, 0.0f, 100.0f, 3,
nNumPoints, &ctrlPoint[0][0]);
       glEnable(GL_MAP1_VERTEX_3);
       glBegin(GL_LINE_STRIP);
       for(int i = 0; i <= 100; i++)
       {
          glEvalCoord1f((GLfloat)i);
       }
       glEnd();
```

```
        glColor3f(0.0f, 0.0f, 0.0f);
        glBegin(GL_POINTS);
        for(int i = 0; i < 4; i++)
        glVertex3f(ctrlPoint[i][0], ctrlPoint[i][1],
ctrlPoint[i][2]);
        glEnd();
        glColor3f(1.0f, 1.0f, 1.0f);
        currentNum = 0;
    }
    glFlush();
}
```

附录 C

附录 C 主要为三维模型检索程序的部分主要代码的实现

```
#pragma region Zernike
//////////////////-------------------------------------
-- Zernike-------------------------
float* CModel_RetrievalDlg::choosepicture(char* src)
{
    float* z_src = new float[35];
    CModel_RetrievalDlg *z_m = new
CModel_RetrievalDlg();
    int index = 0;
    for(int i=2;i<=9;i++)
    {
        //z_m->grayImg = cvLoadImage(src, 1);
        if(i%2==0)
        {
            for(int j=0;j<=i;j+=2)
            {
                z_src[index++] =
z_m->getZernike(src,i,j);
            }
        }
        else
        {
            for(int j=1;j<=i;j+=2)
            {
                if(i==9&&j==5)
                    break;
                z_src[index++] =
z_m->getZernike(src,i,j);
            }
        }
    }
    if(z_m!=NULL)
    delete z_m;
    return z_src;
}
float* CModel_RetrievalDlg::normalize_Zernike(float
*z_modes)//Zernike归一化
{
```

```
        float num=0;
    float* z_image = new float[35];
   for(int i=0;i<35;i++){
        //cout<<z_modes[i]<<"   ";
        num+=z_modes[i];
    }
        for(int i=0;i<35;i++){
        z_image[i] = z_modes[i]/num;

    }
    return z_image;
}
float CModel_RetrievalDlg::getZernike(char* src, int
n, int m)
{
    int pass = Img2Gray(src);
  if(!pass)
     return -1;
   int depth     = 0;
  int nChannels  = 0;
  nChannels     = grayImg->nChannels;
  depth = grayImg->depth;
  switch(depth)
  {
        case   IPL_DEPTH_8U:
Caculate_8_Zernike(n,m);   break;
        case   IPL_DEPTH_32F:
Caculate_32_Zernike(n,m);  break;
        default:    break;
  }
  ClearOpenCV();
  return  Z_mode;
}
float CModel_RetrievalDlg::get_8_XYValue(int x,int y)
{
    int height     = grayImg->height;
   int widthStep   = grayImg->widthStep;
    char*  Data    = grayImg->imageData;
   uchar  c_value =  ((uchar *)(Data+x*widthStep))[y];
    float  value = (float)c_value;
   return value;
}
float CModel_RetrievalDlg::get_32_XYValue(int x,int
```

```
y)
{
    int height     = grayImg->height;
   int widthStep   = grayImg->widthStep;
    char*  Data    = grayImg->imageData;
   float  value =  ((float *)(Data+x*widthStep))[y];
   return value;
}
int CModel_RetrievalDlg::Img2Gray(char* src)
{
    if((grayImg = cvLoadImage(src, 1)) != 0   )
        {
            return 1;
        }
    return 0;
}
void CModel_RetrievalDlg::ClearOpenCV(void)
{
    if(oriImg!=NULL){
       cvReleaseImage( &oriImg  );
        oriImg = NULL;
    }
    if(grayImg!=NULL){
       cvReleaseImage( &grayImg );
        grayImg = NULL;
    }
}
void CModel_RetrievalDlg::Caculate_8_Zernike(int
n,int m)
{
   int height     =  grayImg->height;
   int widthStep   =  grayImg->widthStep;
    float N        =  MinP(height,widthStep);
    float N2       =  N/2;
    float Rpqr_C =0;
    float Rpqr_S =0;
    for(float r=1;r<N2;r++)
    {
       float temp_C = 0;
        float temp_S = 0;
            for(float s=1;s<=8*r;s++)
        {
             float xy_v = get_8_XYValue(r,s);
```

```
            temp_C = temp_C + cos((PI*m*s)/(4*r))*xy_v;
            temp_S = temp_S + sin((PI*m*s)/(4*r))*xy_v;
        }
     float Rpqr =     zernikeR(n,m,(2*r)/N);
     Rpqr_C =Rpqr_C  +    temp_C* Rpqr;
    Rpqr_S = Rpqr_S  +    temp_S* Rpqr;
     }
     Cnm = Rpqr_C*(2*n+2)/pow(N,2);//Z的实部
    Snm = Rpqr_S*(2*n+2)/pow(N,2);//Z的虚部
         float l_c = pow(Cnm,2);
     float l_s = pow(Cnm,2);
     float l_p = l_c + l_s;
    Z_mode = pow((float)l_p,(float)0.5);
}
void CModel_RetrievalDlg::Caculate_32_Zernike(int
n,int m)////////////
{
    int height      =  grayImg->height;
    int widthStep   =  grayImg->widthStep;
     float N        =  MinP(height,widthStep);
     float N2       =  N/2;
     float Rpqr_C =0;
     float Rpqr_S =0;
     for(float r=1;r<N2;r++)
     {
        float temp_C = 0;
         float temp_S = 0;

         for(float s=1;s<=8*r;s++)
         {
              float xy_v = get_32_XYValue(r,s);
                        temp_C+=
cos((PI*m*s)/(4*r))*xy_v;
            temp_S+= sin((PI*m*s)/(4*r))*xy_v;
         }
          float    Rpqr   =    getRpqr(n,m,(2*r)/N);
     Rpqr_C =Rpqr_C  +    temp_C* Rpqr;
    Rpqr_S = Rpqr_S  +    temp_S* Rpqr;
     }
     Cnm = Rpqr_C*(2*n+2)/pow(N,2);
    Snm = Rpqr_S*(2*n+2)/pow(N,2);
     Z_mode =
pow((float)pow(Cnm,2)+pow(Cnm,2),(float)0.5);
```

```
}
float CModel_RetrievalDlg::getRpqr(float p,float
q,float r)
{
      float Rpqr = 0;
     float Bppp=1;
     int times   = (p-q)/2;
    int numbers = times+1;
    float  Bpqp  =   pow((p+q)/(p-q+2),times);
    float* Bpqk = new float[numbers];
       Bpqk[0] = Bpqp;
     int k=(int)p;
      int t=(int)p;
          for(int i=0;i<(numbers-1);i++)
      {
         float  coeff = ((k+q)*(k-q))  /  ((k+p)*(p-
k+2));
          Bpqk[i+1]   =  (-1)*(Bpqk[i])*coeff;
          k=k-2;
      }
      int temp = numbers-1;
          for(k=(int)q;k<=t;k=k+2)
      {
          Rpqr = Rpqr + (Bpqk[temp])*(pow(r,k));
         temp--;
      }
      delete[]  Bpqk;
     float  a = Rpqr;
     float  b = Rpqr;
      return Rpqr;
}
double CModel_RetrievalDlg::factorial(long n)
{
    if(n < 0)
        return(0.0) ;
    if(n == 0)
        return(1.0) ;
    else
        return(n * factorial(n-1)) ;
}
float CModel_RetrievalDlg::zernikeR(int n, int l,
float r)
{
```

```
  int m ;
  float sum = 0.0 ;
    if( ((n-l) % 2) != 0 )
    {
      cout <<"zernikeR(): improper values of n,l\n" ;
      return(0) ;
    }
    for(m = 0; m <= (n-l)/2; m++)
    {
      sum += (pow((float)-1.0,(float)m)) *
( factorial(n-m) ) /
     ( factorial(m) * (factorial((n - 2*m + l) / 2)) *
       (factorial((n - 2*m - l) / 2)) ) *
     ( pow((float)r, (float)(n - 2*m)) );
    }

  return(sum) ;
}
```