**三维扫描建模技术的研究与其在古文物保护方面的应用——结题报告**

****

**097082 甘仕琦**

**097062 饶司君**

**092988吴子奇**

三维扫描建模技术的研究与其在古文物保护方面的应用

## 1、引言

古文物集文化、历史、艺术价值于一体，是古代劳动人民智慧的结晶，是几千年文化发展的积淀。随着社会经济的高速发展，人类对祖先留给自己的古文物遗产的保护越来越重视。古文物测绘作为古文物遗产保护的基础性工作具有深远的意义，随着计算机技术的快速发展和科学技术的不断进步，将现实世界的实体信息快速地转换为计算机可以识别处理的数据已经不再是人类的梦想。科技的创新，不断涌现的新技术为空间数据采集提供了各种各样的新方法和新手段，推动三维空间数据获取向着集成化、实时化、动态化、数字化和智能化的方向发展。三维激光扫描技术就是这个信息获取时代的产物，该技术作为获取空间数据的有效手段，能够快速的获取反映客观事物实时、动态变化、真实形态特性的信息。以至有人称三维激光扫描技术是继GPS技术以来在空间数据获取领域的又一次技术革命。三维激光扫描技术又称“实景复制技术”，能够完整并高精度地重建扫描实物及快速获得原始测绘数据。该技术可以真正做到直接从实物中进行快速的逆向三维数据采集及模型重构，无需进行任何实物表面处理，其激光点云中的每个三维数据都是直接采集目标的真实数据，使得后期处理的数据完全真实可靠。由于技术上突破了传统的单点测量方法，其最大特点就是精度高、速度快、逼近原形，是目前国内外测绘领域研究关注的热点之一。

1916年，爱因斯坦提出了受激辐射理论，首次提出了激光的概念。直到1958年12月15日，美国《物理评论》杂志发表的题为《红外和光的微波激射》论文，宣告了激光的产生。1959年贝尔实验室氦原子的能量激发氖原子，提出了氦氖激光器的原理。1960年，美国休斯研究室的梅曼用固体的红宝石做实验，研制成功了世界上第一台红宝石激光器。1961年9月，中国第一台激光器研制成功。伴随着激光器的发展，激光在诸多领域有着广泛的应用。如:机械零件的激光切割、汽车车身的激光焊接、汽车发动机汽缸内表面的热处理、激光微管手术，激光美容、激光全息摄影、激光雷达、激光测距、激光雕刻、激光武器……

激光技术在大地测量方面的应用以瑞士OPTON公司1968年生产的世界上第一台全站仪ELTA-14为标志。三维激光扫描技术的研究和应用开始于1998年，以瑞士徕卡CYRAX2400为代表。截至2002年10月国内尚没有该类产品的购买和应用。此后，武汉大学研制成功了LD激光扫描测量系统，作为国内第一台三维激光扫描系统主要应用于堆体的扫描和体积计算。

在古建筑文物中，由于中国古代的形态和结构纷繁复杂，传统的测量方法固有的局限性导致无法准确、详细地表达古文物的真实面貌，以三维激光扫描技术为代表的现代测绘技术恰恰弥补了传统测绘方法的不足。为此，借助中国和意大利政府的合作协议，故宫博物院2003年首先由意大利方面引入三维激光扫描技术用于故宫古建筑文物的扫描测绘。其后北京建筑工程学院及清华大学先后购买了相关的扫描系统用于古文物的测绘和研究。由于种种原因，截至目前关于三维扫描技术在古文物测绘中的应用并不令人满意，虽然发表了一些文章，但也仅仅限于基本原理的描述。归纳起来，主要存在以下几个方面的问题:

（1）对新技术缺乏必要的认识

三维扫描技术作为一个新近出现的测绘手段，大多数人并不了解。没有接触过三维扫描的专业测量人员也仅了解其基本原理，对于系统的性能、使用过程中所能达到的精度以及后续的处理等细节问题缺乏了解。与此同时，设备标称的mm级精度是有条件的，实际工作中往往达不到。目前三维扫描系统没有统一的技术标准，不同厂家采用各自的方式去描述产品的性能和特点，各种产品间没有统一的评价指标，也没有统一认可的检定标准。这就导致使用人员在扫描中无法把握扫描的质量，迫切需要从三维激光扫描的原理出发，揭示扫描的精度和扫描的质量，尤其对于三维扫描技术在古建筑测绘中的应用和转化问题需要作进一步深入的研究。

（2）专业间缺乏交叉

三维扫描技术属于测量领域的新技术，学习和使用该技术的往往是测量专业的人员，测量专业的人员一般不具备古建筑测绘的知识，不了解古建筑测绘的特点和特殊需求，理解和使用新技术总是从本专业的角度出发，往往难以满足古建筑测绘的要求。而从事古建筑测绘的建筑专业人员又没有精力去系统学习和掌握相关技术，对新技术的理解有偏差，往往不加任何分析直接利用测量人员提供的测量数据，由此得出错误的结论。典型事例如发表在《故宫博物院院刊》2007年第二期的《佛光寺东大殿实测数据解读》一文，就错误的理解了三维扫描测绘的精度，直接利用了基于点云的亚毫米级量测数据，在此基础上错误地推翻了前人揭示出的包含在中国古建筑中的法式，得出了用全面测绘取代法式测绘的错误结论。由于专业间缺乏交叉导致的使用中存在的误区，妨碍了新技术、新设备面向文物保护领域的有效转化和合理应用，亟需从古建筑测绘的角度去理解和分析作为测量新技术的三维激光扫描技术的内涵和使用中的优劣，纠正和预防在古文物测绘应用中可能出现的误区。

尽管国内在使用三维扫描技术还存在各种各样的问题，但毫无疑问的是，三维扫描技术已经在中国的测绘领域占领了一席之地，对于三维扫描技术的应用也开始越来越成熟。一些常识性的问题已经很少出现在测绘过程中。三维激光扫描技术可以广泛应用于工程建设中与物体三维测量相关的众多领域，包括：水利工程施工测量，可替代传统摄影测量方式进行大坝全区测图，速度快、精度高，在地形复杂地区更显优越性；水利工程大坝、地下厂房等大型建筑物测量维护与仿真、位移监控和外观结构三维建模，做设计与维护分析；水利工程开挖容积测量与应用；高边坡变形监测与快速辅助地质编录，稳定性分析计算的地形数据获取等；危险地区或目标物的遥控测量；土木施工实时测量；隧道收方，变形监测：其它数据源的复合等等。

国内对于三维扫描技术的发展和研究着重于对大型物体的扫描，却很少将三维扫描技术应用到小型物体的数字重现中，如对小型瓷器进行三维建模，以图在瓷器破坏后进行恢复。古代文物亟需进行三维建模备份，而在古文物中，小型文物所占的比例丝毫不小于大型文物。由于，扫描物体较小，要求精度较高，所以按照扫描大型物体时搬动测站的方法就显得累赘和不必要。故我们的上海市创新项目旨在寻找一种新的方法，使得三维扫描技术能够更好地利用在古代文物的保护当中，使得古代文物能够高精度地还原到电子设备当中。

要将分片的点云拼接成为整体，点云配准技术是不可缺少的。20世纪80年代，许多学者对点云配准进行了大量研究 其中，贡献最为突出的是Besl P.J. 等人提出的迭代最近点算法，该方法通过寻找两个点集的对应匹配点之间的关系，计算两个点集的变换参数，以满足给定的收敛精度，最终求得两个点集之间的平移和旋转参数，来完成配准过程。传统ICP算法存在的问题是，将实际情况太过理想化，在现实扫描过程中，两个点集完全相同是比较难做到的，同时，点云配准计算速度对于现有的计算机也是一个很大的挑战。现在许多算法都是在ICP算法的基础上进行改进。基于以上情况，本项目给出了一种改进的扫描方式和对应的改进的ICP算法，并且通过具体实例验证了算法思路的有效性。

## 2、项目背景

本次创新项目采用的三维扫描仪器是同济大学大学购买的FARO Laser Scanner Focus3D。FARO Laser Scanner Focus3D是一款高速三维扫描仪，可用于详细的测量和三维建档，它拥有触摸操作屏，利用激光和照相技术仅在几分钟内即可得到数百万个三维彩色数据点，从而生成有关复杂环境的非常详细的三维图像和几何形状，对已有状态进行精确的数字化再现。可以说用于本次创新项目的研究再合适不过了。

而考虑到古文物的大小、种类以及其珍贵性，我们决定用瓷器来代替。为了能够凸显本次项目扫描方法的优越性，我们采用了青花瓷来进行模拟。青花瓷个头不大，结构丰富，且表面平滑，方便对建模结果进行评价。结果证明，确实起到了很好的效果。

## 3、扫描方法的改进

### 3.1 传统的扫描方法

由于我们所关心的三维信息在扫描过程中难以避免地会出现部分遮挡的情况，这就要求对扫描仪和被扫描物体的相对位置关系进行认为调整。

不难看出，在固定待扫描物（古文物）的前提之下，如果仅仅是要求获取全部的点云数据，测站的摆放方式可以是多样的。如果测站被放在离文物无穷远处，则仅仅需要从正反两面分别取点，进行两次扫描。显然这仅仅是理想状况，而且测站和目标距离太远会导致精度上无法满足。所以在常规的测量中，一般可采用三次扫描来取得全部的点云数据。

确定了测量次数之后，只要测站摆放角度、距离的选择合适，不难保证获取关于外形的所有点云信息，并且都有足够的拼接长度，扫描方式的示意图见图一。之后只要将这些点云导出到计算机，借助ICP算法就可以还原出文物的三维形状。



图1 传统扫描方式

### 3.2 改进后的扫描方式

传统的扫描方式中，测站的摆放很大程度上是依靠测量人员的经验。扫描仪摆放在哪几个角度，距离待扫描物体多远时，可以保证一个良好的扫描效果，这些都没有定性的规律可以遵循。另外，三维扫描仪本身并不像经纬仪、全站仪一样能够精确对中，所以即使有能力从理论上规范测站的摆放方式，在实际操作中也是很难实现的。

虽然在测站任意摆放的情况下也能够顺利得到点云，但是不可否认的是，如果可以将摆放的位置关系规范化，那么后期点云数据处理的工作量将会减轻，带来的好处将是拼接精度和拼接效率的提高，这在对尺寸小、数量大的古文物进行扫描归档时的意义是不言而喻的。

为了达到规范化测站和物体相对位置的目的，可以将思路从搬动测站的方法中转移出来，充分利用古文物相对于古建筑尺寸小的特点，采用旋转古文物的方法来进行扫描。此时的相对位置控制就变得十分方便：由于测站和待扫描物的位置都没有移动，所以此时的距离是恒定不变的常数，而角度又能在合适的方法之下得到精确控制。

在实现精确旋转方面，本项目借助了全站仪的测角功能。通过在全站仪的顶部加上一个经过设计加工的平台，将全站仪转化成一个扫描专用载物台，它不但能够自由旋转，还能精确读出旋转角度，这就为后期点云数据处理铺平了道路。

扫描的过程并不复杂，在全站仪安放好之后，将铁盘固定上去。整平之后，便可以将青花瓷放置上去。将扫描仪放置在合适的位置，调整好合适的精度之后，FARO Laser Scanner Focus3D便能自动扫描，收集点云数据。

如下图二所示，三次旋转，三次扫描之后，能够收集到三组点云数据。由于扫描物体的旋转中心轴无法确定，所以不能直接将这三组点云数据经过简单旋转后进行拼凑。为此我们对旋转中心轴的确定方法做了很长一段时间的研究，比如用靶标球确定旋转轴，但最终得到的结果都不甚理想。那可否在无法确定旋转中心轴的情况下，进行点云数据的拼接呢？从不同视角扫描得到的每一幅点云数据都是相对于自己的扫描空间坐标系而言的，而如何将不同视角下的点云数据统一到一个坐标系统中？项目小组成员在查阅了大量文献资料后，决定采用一种三维激光点云配准算法——ICP算法。



图2 改进后的扫描方式

## 4、ICP算法的改进

### 4.1 传统的ICP算法

能够使不同的坐标下的点云数据合并到同一个坐标系统中，首先是找到一个可用的变换，配准操作实际上是要找到从坐标系coord1到坐标系coord2的一个刚性变换。如果用一个3× 3的旋转矩阵R和一个3维平移向量t来描述这个变换，对于coord1中的任意一点p设其坐标系coord1和坐标系coord2中的坐标值分别为和，都满足如下条件：

ICP算法用来解决多视点云间的对齐问题，前面已经提到ICP算法的由来，现在介绍一下ICP算法的实现过程，ICP算法本质上是基于最小二乘法的最优配准方法。该算法重复进行选择对应关系点对，计算最优刚体变换这一过程，直到满足正确配准的收敛精度要求 多视点云对齐的数学定义可以这样描述：给定两个来自不同坐标系的三维数据点集，找到两个点集空间的变换关系，使得两个点集能统一到同一坐标系统中，即配准过程假定用

表示第一个点集，并用

表示第二个点集，两个点集的对其配准转换为使下列目标函数最小。

（1）

ICP算法的目的是要找到待配准点云数据与参考点云数据之间的旋转参数R和平移参数t，使得两点集数据之间满足某种度量准则下的最优匹配。假设给定两个三维点集和，ICP方法的配准步骤如下所示：

第一步，计算中的每一个点在点集中的对应最近点；

第二步，求得使上述对应点对平均距离最小的刚体变换，求得平移参数和旋转参数；

第三步，对使用上一步求得的平移和旋转参数，得到新的变换点集；

第四步，如果新的变换点集与参考点集满足（1）式目标函数要求，即两点集的平均距离小于某一给定阈值，则停止迭代计算，否则新的变换点集作为新的继续迭代，直到达到目标函数的要求。

ICP算法是一种迭代算法，具有很高的匹配精度，由上述过程可知，该算法存在计算量较大的缺点，同时还可能使得迭代过程无法收敛到全局最优解。

### 4.2 改进后的ICP算法

在对扫描方式进行了改进后，ICP算法也可以做出相应的调整，而最终结果显示，这种调整大大简化了ICP算法拼接过程的计算思路和计算量。

从上面的介绍可以看出，原始的ICP算法的基本思路是不停地寻找合适的R和t，通过足够多次的迭代，最终实现拼接。但在新的扫描方式的帮助下，旋转角度能够被精确控制，即R成为了已知量，此时（1）所示的函数仅仅包含t一个未知量，即

（2）

而t本身是一个代表位移的向量

在消除了变量R之后，t(x)，t(y)，t(z)成为三个独立变量，（2）式可以拆分成下面的形式

由上面三个等式的左边求导等于0立刻可以得到

之后再进行反复迭代，具体方法参照传统ICP算法。

## 5、点云拼接结果

以一个圆柱形笔筒为对象，实现了基于改进的扫描方法、改进的ICP算法的点云数据拼接，以及后续的简单建模处理。

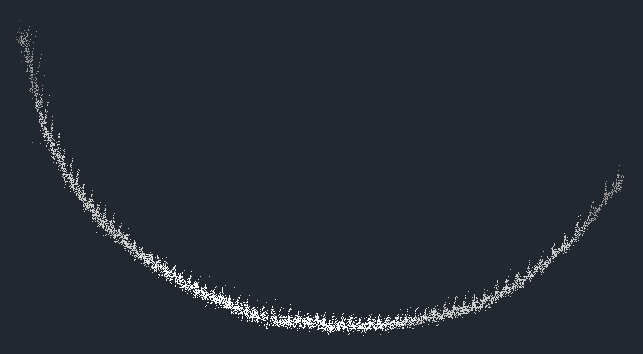


图3 原始数据1/3

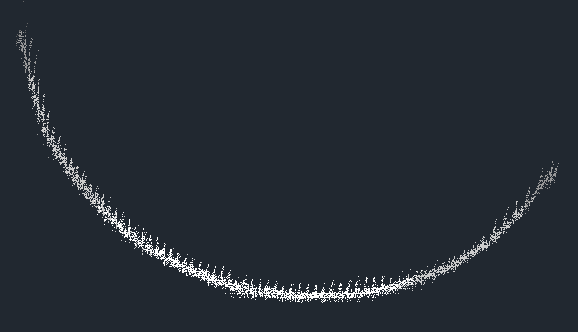


图4 原始数据2/3

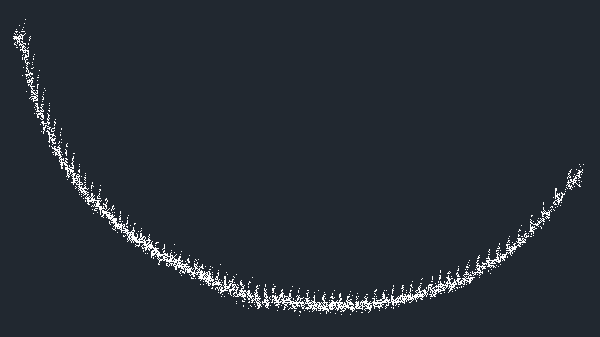


图5 原始数据3/3

上面三幅图显示了拼接前的点在AutoCAD中以俯视图显示的情况，由于是一个形状大致规则的圆柱形的笔筒，所以三次扫描得到的点云数据的形状是很接近的，但仔细观察依然能够看出细微差别。

下图6显示的是经过ICP拼接后的效果。可以看出，除去少量扫描时由环境因素导致的点云遗漏，数据的拼接效果还是能够令人满意的。

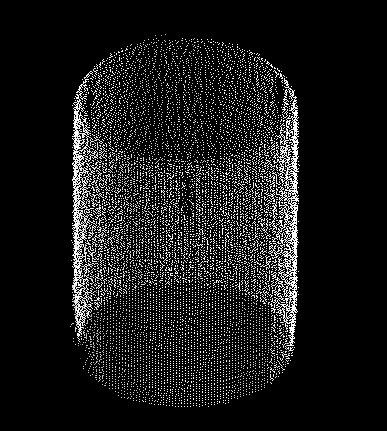


图6 ICP拼接后的点云

## 6、小结

本项目以对古文物扫描为依托，对三维扫描过程和ICP算法做了针对性优化。通过借助全站仪的测角功能，制作了一个专用载物台，实现了扫描仪不搬动，被扫描物精确旋转的扫描方式，并在这种扫描方式的基础上，对ICP算法进行了改进，去掉了迭代计算中的旋转角未知量，大大简化了计算，缩短了迭代时间，同时也提高了拼接的精度。随后在理论基础上，通过C++程序，以青花瓷笔筒为对象，实现了从扫描到点云拼接的一系列设想，有力地证明了这一扫描方法的正确性、优越性，以及数据处理程序的稳定，有效。

附：ICP程序关键部分

***Main.cpp***

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include "dl\_dxf.h"

#include "dl\_creationadapter.h"

#include "Calculator.h"

#include "test\_creationclass.h"

#define kNumber 500

#define kIterative 30

WUDXFParser\* parseDXF(char\* file);

void writeToDXF(DataCloudSet \*set, DataCloudSet \*set2, DataCloudSet \*set3);

void writeVertex(DL\_WriterA \*dw, DL\_Dxf \*dxf, Vertex \*vertex);

int main(int argc, char\*\* argv) {

// Step 1: Read into vertex

WUDXFParser \*firstPart = parseDXF("0004.dxf");

WUDXFParser \*secondPart = parseDXF("0005.dxf");

WUDXFParser \*thirdPart = parseDXF("0006.dxf");

// Step 2: Rotate 120

DataCloudSet \*set1 = firstPart->getCloudSet();

DataCloudSet \*set2 = secondPart->getCloudSet();

DataCloudSet \*set3 = thirdPart->getCloudSet();

for (int i = 0; i < set2->getCount();i ++)

set2->getVertexAt(i)->rotate120();

for (int i = 0; i <set3->getCount();i++)

{

set3->getVertexAt(i)->rotate120();

set3->getVertexAt(i)->rotate120();

}

//////Step 2: Calculate nearest point

Vertex \*\*subset1 = set1->subSet(0,kNumber);

Vertex \*\*subset2 = set2->subSet(0,kNumber);

int i = 0;

while (i < kIterative)

{

//Calculator::findNearestVertex(subset1, thirdPart->getCloudSet(),kNumber);

Calculator::findNearestVertex(subset2, firstPart->getCloudSet(),kNumber);

//// Step 3:Calcalate Move smallest distance

//Calculator::calculateMinmiumDistance(subset1, kNumber, set1);

Calculator::calculateMinmiumDistance(subset2, kNumber, set2);

i ++;

}

writeToDXF(set1, set2, set3);

// Delete

delete []subset1;

delete []subset2;

delete firstPart;

delete secondPart;

delete thirdPart;

//delete testPart

return 0;

}

void writeVertex(DL\_WriterA \*dw, DL\_Dxf \*dxf, Vertex \*vertex)

{

//cout << "Write Point "<< vertex->getX() << " " << vertex->getY() << " " << vertex->getZ() << endl;

dxf->writePoint(

\*dw,

DL\_PointData(vertex->getX(),

vertex->getY(),

vertex->getZ()),

WUDXFAttribute("mainlayer", 7, -3, "BYLAYER")

);

}

WUDXFParser\* parseDXF(char\* file) {

// Load DXF file into memory:

std::cout << "Reading file " << file << "...\n";

WUDXFParser\* creationClass = new WUDXFParser();

DL\_Dxf\* dxf = new DL\_Dxf();

if (!dxf->in(file, creationClass)) {

std::cerr << file << " could not be opened.\n";

return NULL;

}

delete dxf;

return creationClass;

}

void writeToDXF(DataCloudSet \*dataCloud, DataCloudSet \*set2, DataCloudSet \*set3) {

DL\_Dxf\* dxf = new DL\_Dxf();

DL\_Codes::version exportVersion = DL\_Codes::AC1015;

DL\_WriterA\* dw = dxf->out("result.dxf", exportVersion);

if (dw==NULL) {

printf("Cannot open file 'test.dxf' \

for writing.");

}

// Header

dxf->writeHeader(\*dw);

dw->dxfString(9, "$INSUNITS");

dw->dxfInt(70, 4);

dw->dxfString(9, "$DIMEXE");

dw->dxfReal(40, 1.25);

dw->dxfString(9, "$TEXTSTYLE");

dw->dxfString(7, "Standard");

dw->dxfString(9, "$LIMMIN");

dw->dxfReal(10, 0.0);

dw->dxfReal(20, 0.0);

dw->sectionEnd();

dw->sectionTables();

dxf->writeVPort(\*dw);

dw->tableLineTypes(25);

dxf->writeLineType(\*dw, DL\_LineTypeData("BYBLOCK", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("BYLAYER", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("CONTINUOUS", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("ACAD\_ISO02W100", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("ACAD\_ISO03W100", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("ACAD\_ISO04W100", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("ACAD\_ISO05W100", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("BORDER", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("BORDER2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("BORDERX2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("CENTER", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("CENTER2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("CENTERX2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DASHDOT", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DASHDOT2", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("DASHDOTX2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DASHED", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DASHED2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DASHEDX2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DIVIDE", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DIVIDE2", 0));

dxf->writeLineType(\*dw,

DL\_LineTypeData("DIVIDEX2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DOT", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DOT2", 0));

dxf->writeLineType(\*dw, DL\_LineTypeData("DOTX2", 0));

dw->tableEnd();

int numberOfLayers = 2;

dw->tableLayers(numberOfLayers);

dxf->writeLayer(\*dw,

DL\_LayerData("0", 0),

WUDXFAttribute(

std::string(""), // leave empty

DL\_Codes::black, // default color

100, // default width

"CONTINUOUS")); // default line style

dw->tableEnd();

dxf->writeStyle(\*dw);

dxf->writeView(\*dw);

dxf->writeUcs(\*dw);

dw->tableAppid(1);

dw->tableAppidEntry(0x12);

dw->dxfString(2, "ACAD");

dw->dxfInt(70, 0);

dw->tableEnd();

dxf->writeDimStyle(\*dw,

1,

1,

1,

1,

1);

dxf->writeBlockRecord(\*dw);

dxf->writeBlockRecord(\*dw, "myblock1");

dxf->writeBlockRecord(\*dw, "myblock2");

dw->tableEnd();

dw->sectionEnd();

dw->sectionBlocks();

dxf->writeBlock(\*dw,

DL\_BlockData("\*Model\_Space", 0, 0.0, 0.0, 0.0));

dxf->writeEndBlock(\*dw, "\*Model\_Space");

dxf->writeBlock(\*dw,

DL\_BlockData("\*Paper\_Space", 0, 0.0, 0.0, 0.0));

dxf->writeEndBlock(\*dw, "\*Paper\_Space");

dxf->writeBlock(\*dw,

DL\_BlockData("\*Paper\_Space0", 0, 0.0, 0.0, 0.0));

dxf->writeEndBlock(\*dw, "\*Paper\_Space0");

dxf->writeBlock(\*dw,

DL\_BlockData("myblock1", 0, 0.0, 0.0, 0.0));

dxf->writeEndBlock(\*dw, "myblock1");

dxf->writeBlock(\*dw,

DL\_BlockData("myblock2", 0, 0.0, 0.0, 0.0));

dxf->writeEndBlock(\*dw, "myblock2");

dw->sectionEnd();

dw->sectionEntities();

// write all your entities..

for (int i = 0; i < dataCloud->getCount();i++)

{

writeVertex(dw,dxf,dataCloud->getVertexAt(i));

}

for (int i = 0; i < set2->getCount();i++)

{

writeVertex(dw,dxf,set2->getVertexAt(i));

}

for (int i = 0; i < set3->getCount();i++)

{

writeVertex(dw,dxf,set3->getVertexAt(i));

}

//dxf->writeLine(

// \*dw,

// DL\_LineData(25.0, // start point

// 30.0,

// 0.0,

// 100.0, // end point

// 120.0,

// 0.0),

// WUDXFAttribute("mainlayer", 256, -1, "BYLAYER"));

dw->sectionEnd();

dxf->writeObjects(\*dw);

dxf->writeObjectsEnd(\*dw);

dw->dxfEOF();

dw->close();

delete dw;

delete dxf;

}

Calculator.cpp

#include "Calculator.h"

#include<cmath>

#include "math.h"

#define infinitebig 1000000000

void Calculator::findNearestVertex(Vertex \*\*subset, DataCloudSet \*dataSet, int size)

{

for (int i = 0; i < size;i++)

{

subset[i]->setNeightbour(NULL);

findNearestVertex(subset[i], dataSet);

}

}

void Calculator::findNearestVertex(Vertex \*vertex, DataCloudSet \*dataSet)

{

int closesetIndex = -1;

double closestDistance = infinitebig;

for (int i = 0;i < dataSet->getCount();i++)

{

Vertex \*re = dataSet->getVertexAt(i);

if (re != NULL)

{

float result = GetDistance(vertex, re);

if (result < closestDistance)

{

closestDistance = result;

closesetIndex = i;

}

}

}

vertex->setNeightbour(dataSet->getVertexAt(closesetIndex));

}

float Calculator::GetDistance(Vertex \*a, Vertex \*b)

{

float multiply = pow((a->getX() - b->getX()),2) + pow((a->getY() - b->getY()),2) + pow((a->getZ() - b->getZ()),2);

return sqrt(multiply);

}

void Calculator::calculateMinmiumDistance(Vertex \*\*subset, int size, DataCloudSet \*dataSet)

{

float xSum = 0 , ySum = 0, zSum = 0;

/\*for (int i = 0;i < size;i ++)

{

xSum += fabs(subset[i]->getX() - subset[i]->getNeighbor()->getX());

ySum += fabs(subset[i]->getY() - subset[i]->getNeighbor()->getY());

zSum += fabs(subset[i]->getZ() - subset[i]->getNeighbor()->getZ());

}\*/

/\*xSum /= size;

ySum /= size;

zSum /= size;\*/

for (int i = 0;i < size;i ++)

{

xSum += subset[i]->getX() - subset[i]->getNeighbor()->getX();

ySum += subset[i]->getY() - subset[i]->getNeighbor()->getY();

zSum += subset[i]->getZ() - subset[i]->getNeighbor()->getZ();

}

xSum = getInvertNumber(size, xSum);

ySum = getInvertNumber(size, ySum);

zSum = getInvertNumber(size, zSum);

for (int i = 0;i < dataSet->getCount();i ++) dataSet->getVertexAt(i)->moveBy(xSum, ySum, zSum);

}

float Calculator::getInvertNumber(int size, int parameter)

{

float result = 0;

parameter = -1 \* parameter;

result = parameter / size;

return result;

}

***Vertex.cpp***

#include "vertex.h"

float Vertex::a[3][3] = {{cos120, -sin120, 0}, {sin120, cos120, 0} ,{0 ,0 ,1}};

Vertex::Vertex(float x, float y, float z)

{

xPos = x;

yPos = y;

zPos = z;

neighbourPoint = NULL;

//rotate120();

}

void Vertex::moveBy(float x, float y, float z)

{

xPos += x;

yPos += y;

zPos += z;

}

void Vertex::rotate120()

{

float xOld = xPos;

float yOld = yPos;

xPos = cos120 \* xOld + (-sin120 \* yOld);

yPos = sin120 \* xOld + cos120 \* yOld;

zPos = zPos;

}

***DataCloudSet.cpp***

#include "DataCloudSet.h"

void DataCloudSet::addVertex(Vertex \*vertex)

{

vetex.push\_back(vertex);

}

DataCloudSet::DataCloudSet()

{

}

void DataCloudSet::print()

{

cout << "Count of vector is " << vetex.size() << endl;

}

Vertex \*DataCloudSet::getVertexAt(int i)

{

if (i < 0 || i >= vetex.size())

return NULL;

return vetex[i];

}

Vertex \*\*DataCloudSet::subSet(int start, int count)

{

if (start < 0 || (start + count) >= vetex.size()) return NULL;

Vertex \*\*subset = new Vertex \*[count];

for (int i = start; i < count;i ++)

{

subset[i] = new Vertex();

subset[i] = vetex[i];

}

return subset;

}