POCLUST Fall Research

William K. Bittner

December 16th, 2011

1 Introduction

In the High Energy Physics lab at Florida Tech, under Dr. Hohlmann, research is being conducted using muon tomography for passive detection of High Atomic Number materials using GEM(Gas Electron Multiplier) Detectors. It has been successful so far, being able to produce tomographic images and visualize them using a POCA (Point of closest Approach) and Vx4D for visualization. Vx4D is an in-house developed C++, OpenGL, and Linux package that allows you to visualize the reconstructed POCA points. In the efforts to create a more accurate reconstruction algorithm, POCLUST was created. The POCLUST Algorithm is a density based clustering algorithm developed by Dr. Debasis Mitra. It was designed to be a next step up from POCA (Point of Closest Approach) algorithm that is currently used today in the reconstruction methods.

2 Goals for the Fall

For undergraduate research being done remotely, the following goals were established and completed:

- Receive the old code base from Dr. Mitra, and get it to compile
- Convert the old code base to now work with our new format of data
- Find a work around, since we do not have momentum data in the real data
- Fix anything that needs to be fixed in the old code base
- Build a visualization framework that is capable of visualizing the clusters from the POCLUST output
- Build a visualization framework that is capable of visualizing the original target positions for comparison
- Build an algorithm that would be able to compare the accuracy of the POCLUST output to the actual target locations
- Visualize in Vx4D, run through POCLUST, and compare with target scenario of High statistic Monte Carlo Data
- Visualize in Vx4D, run through POCLUST, and compare with target scenario of real data captured from our station

3 Location of New POCLUST code base

Currently, the latest updates of POCLUST and the Visualization algorithms are kept on the GIT repo. Every major change is then archived as a tar ball, and pushed to the cluster. The location of the latest POCLUST code on the cluster is in /home/wbittner/poclust/latest/ . There are two sections of the code base latest/algo/ and latest/visual/

3.1 algo

This section contains all the code, mostly original, from the pocclust algorithm. It takes raw POCA output, NOT VOXELIZED, as input, in input.txt. You then run ./make.sh, and you will receive statistical output both to the screen and to a file called output.txt. The visualization output is stored in cluster_vis_output.m. This you will move to the visual side of the poclust code base. Also, copy the input.txt to ifile.txt, and run root -l voxelize.c, this will voxelize the raw POCA data for vx4d and store it to ofile.txt, copy this also to the visual side.

3.2 visual

This section contains all the code written for the visualization this fall. Once the poclust algorithm is run in algo, make a new directory in visual/bin/foo_bar and copy cluster_vis_output.m to visual/bin/foo_bar/foo_bar_cluster.dat. Also, copy ofile.txt from the root -l voxelize.c from above to visual/bin/foo_bar/foo_bar_vx4d.dat, this is for the vx4d comparison. Now copy your favorite vx4d config config.dat to visaul/bin/foo_bar/config.dat , and configure it properly for the dimension size of the volume. Finally, open a text editor, and create a file visual/bin/foo_bar/foo_bar_target.dat. This file will hold the target that you want to compare it with. The target format is:

```
volume_x_size volume_y_size volume_z_size
l w h cx cy cz weight object
```

l is the length of the target, w is the width, h is the heigh, cx is the x center, cy, is the y center, and cz is the z center. The weight represents the value from 0 to 1.0 of whatever you want the targets color to represent. POCLUST uses angle variance as of now. There is only one type of object support in POCLUST, and that is a rectangular object, so object is always set to 1.
Finally you run: ./test.sh foo_bar
this will load both the POCLUST display and the voxelized vx4d display

4 The old POCLUST code base

4.1 State of the original code base

When the code base was first received, there was no documentation on how to run the code. There were very little comments through out the code. Variable names gave little or now description to what they were actually used for, and coding style was terrible. There were also quite a few more C files in the directory, that were not needed, or being compiled, I am still not to sure what they

are for. For future work, I highly reccomend that the code base is cleaned up, especially if it is going to be used it the future. If it grows much larger, it will not be maintainable.

4.2 Convert the old code base to work with new data formats

The old POCLUST code base was built to take raw, PRE-POCA, GEANT4 Montecarlo data, and run its own POCA reconstruction on every set of hits before running the clustering algorithm, also it was supplied momentum data. Now, we run POCA reconstruction on the data as soon as we get it, and we share and use it in the post POCA form. By converting POCLUST to use POCA data instead of raw data, it saved a considerable amount of time each run, since each run POCLUST did not have to redo all the POCA reconstruction, it was already done.

4.3 Momentum Data, or lack of

The old POCLUST algorithm required momentum data from the incoming muon. It would receive this data as an output from GEANT4, as it is an option to output the momentum data. POCLUST would then use the momentum data to normalize the scattering angle of them muon, as such:

$$P_{norm} = P_{muon}/P_0 \tag{1}$$

Where $P_0 = 3.0$ and is hard coded in common.h as a # define constant
Since we do not have momentum data from out real muon tomography station yet, all of our real data and Monte Carlo simulation data does not have it either. So to work around this problem, we fixed P_{muon} to 3.0, as if it always had full momentum. We know that this will cause the POCLUST algorithm to be less efficient, but the exact consequences are unknown.

4.4 Error in the variance algorithm

When running POCLUST on the FeHigh, ultra high statistic scenario with just an iron block in the middle of the minamal MTS, it was noted that the variance of the deflection angle was very high. It was about 60. With further inspection, it was noted that the online variance algorithm does not seem to be correct. So it was changed from:

```
double computeVariance(double varold,double unew,double data,double Nnew){
    if(Nnew==1)
        return(0.0);
    double uold,varnew;
    uold= ((unew*Nnew)-data)/(Nnew-1); //calculate the old average
    varnew= ((Nnew-2)*varold + data*data + uold*uold - unew*unew)/(Nnew-1);
    return(varnew);
}
```

To the new on-line algorithm:

```
double computeVariance(double variance, double currentMean, double newTheta,
                       double numberNew)
{
    double newMean, newVariance;
    if(numberNew <= 1) return (0.0);
    newMean = currentMean + ( (newTheta - currentMean) / (numberNew));
    newVariance = ( (numberNew - 1)*variance + (newTheta - newMean)*
                    (newTheta - currentMean)) / (numberNew);
    return newVariance;
}
```

This change in the algorithm brought down the variance to about 20 for the feHigh high statistic Monte Carlo data, which is more of what is expected. This algorithm was also pulled out and tested independently on a set of numbers with a known variance.

5 POCLUST Cluster Visualization framework

After making the above changes to the POCLUST algorithm, the algorithm was run on Monte Carlo data, and results similar to that of before were achieved. This verified that the algorithm is now working again. Upon the completion of the algorithm, to files are created. One that is called output.txt with various statistics, and one that is the output for the visualization. With the current method of visualization, it was set to use matlab, to plot the clusters. For the following reasons, matlab was chosen not to be used:

- For research use, the license costs THOUSANDS of dollars
- Everyone would have to buy a copy
- It cannot be fully customized to what we need
- It is slower than a well written custom visualization suite
- It is not meant for tomographic reconstruction visualization

So in order to be able to visualize the output of POCLUST, a new frame work was needed. The framework can be seperated into the following files.

5.1 main.cpp - The main setup

In main.cpp the following operations are done:

- The arguments are taken from the command line
- The framework for XWindows is setup
- The OpenGL framework is setup

- OpenGL is attached to the XWindows Window
- The initialization of VisualMain.cpp
- The creation of the infinite drawing loop, and the calls to VisualMain's draw function in it

Code listing for main.cpp:

```
#include <GL/glut.h>
#include <string.h>
#include <stdlib.h>
#include <string>
#include <X11/Xlib.h>
#include <X11/keysym.h>
#include <GL/gl.h>
#include <GL/glx.h>
#include <iostream>
#include <stdio.h>
#include "include/visual_main.h"
using namespace std;
static void event_loop(Display *dpy, Window win);
static void key(unsigned char key, int x, int y);
VisualMain *main_vis;
void setupGL();
static void display();
static void reshape(int width, int height);
static void make_window( Display *dpy, const char *name,
                         int x, int y, int width, int height,
                         Window *winRet, GLXContext *ctxRet);
static void idle(void);
int main (int argc, char *argv[]) {
    if (argc != 3 &&  argc != 6) {
        cout << "Usage: pvis OriginalObjects.dat ClusterObjects.dat \n";
        return -1;
    }
    std::string target_file(argv[1]);
    std::string cluster_file(argv[2]);
    main_vis = new VisualMain(target_file,cluster_file);
    if (argc == 6)
    {
        main_vis->setRotation((GLfloat)atoi(argv[3]),
                              (GLfloat)atoi(argv[4]),
                              (GLfloat)atoi(argv[5]));
    }
    Display *dpy;
    Window win;
    GLXContext ctx;
```

```
    char *dpyName = NULL;
    GLboolean printInfo = GL_FALSE;
    for (int i=1; i < argc; i++)
    {
        if(strcmp(argv[i], "-display") == 0)
        {
            dpyName = argv[i+1];
            i++;
        }
        else if ( strcmp(argv[i], "-info") == 0)
        {
            printInfo = GL_TRUE;
        }
    }

    //open the xDisplay
    dpy = XOpenDisplay(dpyName);
    if(!dpy)
    {
        std::cout << "Uh oh, there was an error, cannot open display "<<  dpyName;
    }

    make_window(dpy, "Cluster Visualization", 0, 0, 800, 600, &win, &ctx);
    XMapWindow(dpy, win);
    glXMakeCurrent(dpy, win, ctx);
    reshape(800,600);

    //setup the OpenGL
    setupGL();

    // main loop
    event_loop(dpy,win);

    // now everything is done, so exit
    glXDestroyContext(dpy,ctx);
    XDestroyWindow(dpy, win);
    XCloseDisplay(dpy);

    return EXIT_SUCCESS;

}
static void event_loop(Display *dpy, Window win)
{
    while(1)
    {
        display();
        glXSwapBuffers(dpy,win);
```

```
    }
}
static void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //glPushMatrix();
    main_vis->draw();
    //glPopMatrix();
}


static void key(unsigned char key, int x, int y)
{
    //capsule->getInput()->keyPress(key,x,y);
}

void setupGL()
{
    int ar = 1;
    char *arg[] = {"Cluster Visualization", NULL};
    glutInit(&ar,arg);

    glutKeyboardFunc(key);

    glutIdleFunc(idle);

    glClearColor(0,0,0,1);

    glEnable(GL_DEPTH_TEST);

}

static void idle(void) {
    // Nothing
}
static void make_window( Display *dpy, const char *name,
             int x, int y, int width, int height,
             Window *winRet, GLXContext *ctxRet)
{
   int attrib[] = { GLX_RGBA,
            GLX_RED_SIZE, 1,
            GLX_GREEN_SIZE, 1,
            GLX_BLUE_SIZE, 1,
            GLX_DOUBLEBUFFER,
            GLX_DEPTH_SIZE, 1,
            None };
   int scrnum;
```

```
XSetWindowAttributes attr;
unsigned long mask;
Window root;
Window win;
GLXContext ctx;
XVisualInfo *visinfo;

scrnum = DefaultScreen( dpy );
root = RootWindow( dpy, scrnum );

visinfo = glXChooseVisual( dpy, scrnum, attrib );
if (!visinfo) {
   printf("Error: couldn't get an RGB, Double-buffered visual\n");
   exit(1);
}

/* window attributes */
attr.background_pixel = 0;
attr.border_pixel = 0;
attr.colormap = XCreateColormap( dpy, root, visinfo->visual, AllocNone);
attr.event_mask = StructureNotifyMask | ExposureMask | KeyPressMask;
mask = CWBackPixel | CWBorderPixel | CWColormap | CWEventMask;

win = XCreateWindow( dpy, root, 0, 0, width, height,
            0, visinfo->depth, InputOutput,
            visinfo->visual, mask, &attr );

/* set hints and properties */
{
   XSizeHints sizehints;
   sizehints.x = x;
   sizehints.y = y;
   sizehints.width  = width;
   sizehints.height = height;
   sizehints.flags = USSize | USPosition;
   XSetNormalHints(dpy, win, &sizehints);
   XSetStandardProperties(dpy, win, name, name,
                          None, (char **)NULL, 0, &sizehints);
}

ctx = glXCreateContext( dpy, visinfo, NULL, True );
if (!ctx) {
   printf("Error: glXCreateContext failed\n");
   exit(1);
}

XFree(visinfo);
```

```
   *winRet = win;
   *ctxRet = ctx;
}

static void reshape(int width, int height)
{
   //GLfloat h = (GLfloat) height / (GLfloat) width;
    GLfloat ar = (float) width / (float) height;
   glViewport(0, 0, (GLint) width, (GLint) height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-ar, ar,-1.0, 1.0, 1.0, 2000.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   //gluLookAt(0,0,20,0,0,0,0,1,0);
   main_vis->setup_matrix_for_plot_size();
}
```

5.2 The heart of the display

VisualMain is the heart of the POCLUST clustering display. When it is first initialized by maincpp as mainvis it does the following:

- Creates a new wObject, a class I use to hold common OpenGL functions
- Load the cluster data file's length, width, and height
- Load all the clusters in the file into a vector
- Load the target data file's length, width, and height
- Load all the targets into the vector
- Calls the first (could be any) targets function of get similarity, this calculates the similarity between ALL the targets and ALL the clusters

Header file, visual_main.h:

```
#ifndef VISUALMAIN_H
#define VISUALMAIN_H
#include "wObject.h"
#include "cluster_object.h"
#include "clusterDataFile.h"
#include <string>
#include "wColorMap.h"
```

```
class VisualMain {
    public:
        VisualMain(std::string target_filename, std::string cluster_filename);
        wObject *draw_object; // holds wObject for drawing
        void set_plot_size(GLfloat x_len, GLfloat y_len, GLfloat z_len);
        void setup_matrix_for_plot_size();
        void draw();
        void add_cluster_object(cluster_object *cluster);
        void setRotation(GLfloat xr, GLfloat yr, GLfloat zr);

        wColorMap colorMap; // the color map for the objects

        vector<cluster_object *> cluster_objects;
        vector<cluster_object *> target_objects;
        GLfloat plotSizeX, plotSizeY, plotSizeZ;

        clusterDataFile *theClusterDataFile;
        clusterDataFile *theTargetDataFile;

        // the current rotation amount each refresh of that object
        GLfloat x_rot, y_rot, z_rot;

        // the amount of seperation between the two plots
        GLfloat x_seperation, y_seperation, z_seperation;

};
#endif
```

The visualmain.cpp file:

```
#include "visual_main.h"
#include <string>
#include <iostream>
VisualMain::VisualMain(std::string target_filename, std::string cluster_filename)
{
    draw_object = new wObject();
    plotSizeX = plotSizeZ = 30;
    plotSizeY = 60;
    x_rot = y_rot = z_rot = 0;

    //get cluster database file
    theClusterDataFile = new clusterDataFile(cluster_filename, &cluster_objects);
    theTargetDataFile = new clusterDataFile(target_filename, &target_objects);
    plotSizeX = theClusterDataFile->len_x;
    plotSizeY = theClusterDataFile->len_y;
```

```
    plotSizeZ = theClusterDataFile->len_z;

    y_seperation = z_seperation = 0;
    x_seperation = plotSizeX;


    std::cout << "The number of cluster objects added: " <<
                    cluster_objects.size() << " \n";

    std::cout << "The number of target objects added: " << target_objects.size()
                << " \n";
    GLfloat total_volume = 0;
    GLfloat total_sim = 0; // total similarity between all targets and clusters
    for (unsigned int l=0; l < target_objects.size(); l++)
    {
        total_volume += target_objects.at(l)->volume;
    }
    total_sim = target_objects.at(0)->get_similarity(&target_objects,
                                                     &cluster_objects);
    std::cout << "The total similarity between targets and clusters is: "
              << total_sim*100 << "\n";
    std::cout << "The total target volume is: " << total_volume << "\n";
}
void VisualMain::setRotation(GLfloat xr, GLfloat yr, GLfloat zr)
{
    x_rot = xr;
    y_rot = yr;
    z_rot = zr;
}

void VisualMain::draw()
{
```

The draw function is the function that is called by OpenGL in main.cpp everytime the screen refreshes The glRotatef's below are for rotating the volume. Currently, and third set of parameters can be accepted on the command line to dictate how fast the volume will rotate around each axis

```
    glRotatef(x_rot,1,0,0);
    glRotatef(y_rot,0,1,0);
    glRotatef(z_rot,0,0,1);
```

The for loop below, goes through all the cluster objects, and then maps a color to them according to a color mapping algorithm in wColor.cpp. draw_object is the OpenGL library I built, and drawQuad draws a quad object with the color specified in glColor4f, and takes the cluster_object as a parameter so it knows the size, and the location.

```
    for (unsigned int i=0; i < cluster_objects.size(); i++)
    {
        glColor4f(cluster_objects.at(i)->weight,0,0,1);
        GLfloat the_weight = cluster_objects.at(i)->weight * 100;
        glColor4f(colorMap.R.mapColor(the_weight),
                  colorMap.G.mapColor(the_weight),
                  colorMap.B.mapColor(the_weight),
                  1); // map color from 0 - 100
        draw_object->drawQuad(cluster_objects.at(i));
    }
    // draw the outline to the cluster plots
    draw_object->drawWireOutline(plotSizeX, plotSizeY, plotSizeZ,0,0,0);

    //draw the to the target plots
    draw_object->drawWireOutline(plotSizeX,plotSizeY, plotSizeZ,
                                 x_seperation, y_seperation, z_seperation);

    glTranslatef(x_seperation, y_seperation, z_seperation);
    for (unsigned int j=0; j < target_objects.size(); j++)
    {
        GLfloat the_weight = target_objects.at(j)->weight * 100;
        glColor4f(colorMap.R.mapColor(the_weight),
                  colorMap.G.mapColor(the_weight),
                  colorMap.B.mapColor(the_weight),
                  1); // map color from 0 - 100
        draw_object->drawQuad(target_objects.at(j));
    }
    glTranslatef(-x_seperation, -y_seperation, -z_seperation);

    return;
}
void VisualMain::add_cluster_object( cluster_object *object )
{
    cluster_objects.push_back(object);
    // add the object to the vector of cluster_objects
}
void VisualMain::set_plot_size(GLfloat x_len, GLfloat y_len, GLfloat z_len)
{

}
```

setup_matrix_for_plot_size() will setup the viewing matrix based on the size of the plot. The matrix must be manipulated for changing sizes or you wont be able to see anything. Once you aquire the plot size from the initialization of this class and from the cluster data and target data files, you can call this function.

```
void VisualMain::setup_matrix_for_plot_size()
{
    GLfloat max_size = plotSizeY;
    if ( plotSizeX > plotSizeY ) max_size = plotSizeX;
    if ( plotSizeZ > plotSizeY )
    {
        if (plotSizeZ > max_size) max_size = plotSizeZ;
    }

    gluLookAt(plotSizeX / 2.0, plotSizeY / 2.0, max_size * 2, plotSizeX / 2.0,
             plotSizeY / 2.0, plotSizeZ / 2.0, 0, 1,
              0);
    glRotatef(1,0,0,x_rot);
    glRotatef(0,1,0,y_rot);
    glRotatef(0,0,1,z_rot);
}
```

5.3 clusterDataFile.cpp - Loads all the data

This file is pretty straight forward, it loads the dimensions of the plotting area, then begins to load all the data for each of the clusters, stores them in a STL vector and returns it.

```
#include <stdio.h>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <vector>
#include <string>
#include "clusterDataFile.h"
using namespace std;
/*
* File format: l w h cx cy cz weight object
*
*
*/
clusterDataFile::clusterDataFile(std::string filename,
                                 std::vector<cluster_object*> *the_objects)
```

```
 {
    initVars();
    theFile = filename;
    objects = the_objects;
    fileOpenResult = loadFile();
}

void clusterDataFile::initVars() {

}

int clusterDataFile::loadFile() {
    dataFileStream.open(theFile.c_str());
    float l_x, l_y, l_z, c_x, c_y, c_z, weight;
    float object;
    if(!dataFileStream.is_open()) {
        cout << "Invalid data file: " << theFile << endl;
        exit(-1);
    }
    dataFileStream >> len_x >> len_y >> len_z;
    for (int p =0; dataFileStream >> l_x >> l_y >> l_z >> c_x >> c_y >> c_z
                                  >> weight >> object; p+=1)
    {
        objects->push_back(new cluster_object(l_x, l_y, l_z, c_x, c_y,
                                              c_z, weight, object));
    }
    return 0;
}
```

5.4 wObject.cpp - All my helper OpenGL Functions

This is the file that I have created over the times that has a lot of useful functions I use when working in OpenGL, like drawing axis, plot outlines, tick marks, etc

```
#include "wObject.h"
//#include <cluster_object.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <cstring>
#include <string>
#include <cstdlib>
#include <stdio.h>
#include <fstream>
```

```
wObject::wObject()
{
    //ctor
    quadratic = gluNewQuadric();
    gluQuadricNormals(quadratic, GLU_NONE);
    gluQuadricDrawStyle(quadratic,GLU_LINE);
}
void wObject::drawQuad(cluster_object *obj)
{
    if (obj == NULL) {
        fprintf(stderr, "Bad object given to draw quad\n");
        return;
    }

    drawQuad((GLfloat)obj->center_x, (GLfloat)obj->center_y,
             (GLfloat)obj->center_z,
             (GLfloat)obj->len_x, (GLfloat)obj->len_y,(GLfloat)obj->len_z);

}
void wObject::drawQuad(GLfloat x, GLfloat y, GLfloat z,
                       GLfloat l_x, GLfloat l_y, GLfloat l_z ) {
  glBegin(GL_QUADS);
  GLfloat xvW, yvW, zvW;
  /*if(axis == 'x') xvW = l;
  if(axis == 'y') yvW = l;
  if(axis == 'z') zvW = l;*/

  xvW = l_x;
  yvW = l_y;
  zvW = l_z;

  //top
  glVertex3f( x + xvW, y + yvW,z - zvW); // Top Right Of The Quad (Top)
  glVertex3f(x - xvW, y + yvW,z - zvW); // Top Left Of The Quad (Top)
  glVertex3f(x - xvW, y + yvW, z + zvW); // Bottom Left Of The Quad (Top)
  glVertex3f( x + xvW, y + yvW, z + zvW); // Bottom Right Of The Quad

  // bottom of cube
  //glColor3f(1.0f,0.5f,0.0f); // Set The Color To Orange
  glVertex3f( x + xvW,y - yvW, z + zvW); // Top Right Of The Quad (Bottom
  glVertex3f(x - xvW,y - yvW, z + zvW); // Top Left Of The Quad (Bottom)
  glVertex3f(x - xvW,y - yvW,z - zvW); // Bottom Left Of The Quad (Bottom)
  glVertex3f( x + xvW,y - yvW,z - zvW); // Bottom Right Of The Quad (Bottom)

  // front of cube
  //glColor3f(1.0f,0.0f,0.0f); // Set The Color To Red
  glVertex3f( x + xvW, y + yvW, z + zvW); // Top Right Of The Quad (Front)
```

```
  glVertex3f(x - xvW, y + yvW, z + zvW); // Top Left Of The Quad (Front)
  glVertex3f(x - xvW,y - yvW, z + zvW); // Bottom Left Of The Quad (Front)
  glVertex3f( x + xvW,y - yvW, z + zvW); // Bottom Right Of The Quad

  // back of cube.
  //glColor3f(1.0f,1.0f,0.0f); // Set The Color To Yellow
  glVertex3f( x + xvW,y - yvW,z - zvW); // Top Right Of The Quad (Back)
  glVertex3f(x - xvW,y - yvW,z - zvW); // Top Left Of The Quad (Back)
  glVertex3f(x - xvW, y + yvW,z - zvW); // Bottom Left Of The Quad (Back)
  glVertex3f( x + xvW, y + yvW,z - zvW); // Bottom Right Of The Quad

  // left of cube
  //glColor3f(0.0f,0.0f,1.0f); // Blue
  glVertex3f(x - xvW, y + yvW, z + zvW); // Top Right Of The Quad (Left)
  glVertex3f(x - xvW, y + yvW,z - zvW); // Top Left Of The Quad (Left)
  glVertex3f(x - xvW,y - yvW,z - zvW); // Bottom Left Of The Quad (Left)
  glVertex3f(x - xvW,y - yvW, z + zvW); // Bottom Right Of The Quad (Left)

  // Right of cube
  //glColor3f(1.0f,0.0f,1.0f); // Set The Color To Violet
  glVertex3f( x + xvW, y + yvW,z - zvW);         // Top Right Of The Quad (Ri)
  glVertex3f( x + xvW, y + yvW, z + zvW); // Top Left Of The Quad (Right)
  glVertex3f( x + xvW,y - yvW, z + zvW); // Bottom Left Of The Quad (Righ)
  glVertex3f( x + xvW,y - yvW,z - zvW); // Bottom Right Of The Quad (Right)
  glEnd();
  //glPushMatrix();
  //glTranslatef(x,y,z);
  //glutSolidCube(1.0);
  //glPopMatrix();
}
void wObject::drawWireRect(GLfloat beginX, GLfloat beginY, GLfloat beginZ,
                           GLfloat xLen, GLfloat yLen, GLfloat zLen, int nSlices)
{
    wireRect(beginX + (xLen/2), beginY + (yLen/2), beginZ + (zLen / 2), xLen,
             yLen, zLen,nSlices);

}
void wObject::drawWireCylinder(GLfloat centerX, GLfloat centerY, GLfloat centerZ,
                               GLfloat radius, GLfloat height, int nSlices) {


    glPushMatrix();
    glTranslatef(centerX, centerY, centerZ);
    //glutWireCylinder(radius, height, nSlices, nSlices);
    gluQuadricDrawStyle(quadratic,GLU_LINE);
    gluCylinder(quadratic,radius, radius, height, 40, 1);
    //cout << "draw cylinder \t " << radius << "\t" << height << endl;
```

```
    glPopMatrix();
};
#define FALSE 1
#define TRUE 0
int wObject::WindowDump()
{
    ofstream ofile;
   int i,j;
   static int counter = 0; /* This supports animation sequences */
   char fname[32];
   unsigned char *image;
   int width = 1000;
   int height = 1000;

   /* Allocate our buffer for the image */
   if ((image = (unsigned char*)malloc(3*1000*1000*sizeof(char))) == NULL) {
      fprintf(stderr,"Failed to allocate memory for image\n");
      return(FALSE);
   }


   glPixelStorei(GL_PACK_ALIGNMENT,1);

   /* Open the file */
   //if (stereo)
      //sprintf(fname,"L_%04d.raw",counter);
   //else
      sprintf(fname,"C_%04d.raw",counter);
   /*if ((fptr = fopen(fname,"w")) == NULL) {
      fprintf(stderr,"Failed to open file for window dump\n");
      return(FALSE);
   }*/
   ofile.open(fname);

   /* Copy the image into our buffer */
   glReadBuffer(GL_BACK_LEFT);
   glReadPixels(0,0,width,height,GL_RGB,GL_UNSIGNED_BYTE,image);

   /* Write the raw file */
   /* fprintf(fptr,"P6\n%d %d\n255\n",width,height); for ppm */
   for (j=height-1;j>=0;j--) {
      for (i=0;i<width;i++) {
         //fputc(image[3*j*width+3*i+0],fptr);
        ofile << (char)image[3*j*width+3*i+0];
        ofile << (char)image[3*j*width+3*i+1];
        ofile << (char)image[3*j*width+3*i+2];
         //fputc(image[3*j*width+3*i+1],fptr);
```

```
         //fputc(image[3*j*width+3*i+2],fptr);
       }
   }
   //fclose(fptr);
    ofile.close();

   /* Clean up */

   free(image);
   return(TRUE);
}
void wObject::wireRect(GLfloat centerX, GLfloat centerY, GLfloat centerZ,
                       GLfloat xLen, GLfloat yLen, GLfloat zLen, int nSlices)
{


    glPushMatrix();
    glTranslatef(centerX-(xLen/2.0), centerY-(yLen/2.0), centerZ-(zLen/2.0));
    glScalef(xLen,yLen,zLen);
    glutWireCube(1);

    glPopMatrix(); //Push/pop to avoid messing up current matrix
}
void wObject::wireRect2d(GLfloat bLx, GLfloat bLy, GLfloat bLz, GLfloat uLx,
                         GLfloat uLy, GLfloat uLz)
{
    glBegin(GL_LINE_STRIP);

        glVertex3f(bLx, uLy, uLz);
        glVertex3f(uLx, uLy, uLz);
        glVertex3f(uLx, bLy, bLz);
        glVertex3f(bLx, bLy, bLz);


    glEnd();
    glBegin(GL_LINE_STRIP);
        glVertex3f(bLx, bLy, bLz);
        glVertex3f(bLx, uLy, uLz);
    glEnd();
}
void wObject::drawTick(GLfloat xpos, GLfloat ypos, GLfloat zpos,
                       GLfloat len, char axisLenIsOn) {
    glPushMatrix();
    char a = axisLenIsOn;
    GLfloat xb,xe,yb,ye,zb,ze;
    xb = xe = xpos;
    yb = ye = ypos;
```

```
    zb = ze = zpos;
    if(a == 'x' || a == 'X') {
        xb -= len/2;
        xe += len/2;
    }
    if(a == 'y' || a == 'Y') {
        yb -= len/2;
        ye += len/2;
    }
    if(a == 'z' || a == 'Z') {
        zb -= len/2;
        ze += len/2;
    }

    glBegin(GL_LINES);
        glVertex3f(xb,yb,zb);
        glVertex3f(xe,ye,ze);
    glEnd();
    glPopMatrix();
}
void wObject::printString(char* s)
{
   if (s && strlen(s)) {
      while (*s) {
         glutStrokeCharacter(GLUT_STROKE_ROMAN, *s);
         s++;
      }
   }
}
void wObject::printString(string a) {
    for(unsigned int j=0; j < a.size() ; j++) {
        glutStrokeCharacter(GLUT_STROKE_ROMAN, a[j]);
    }
}
/***** Plot Outline Stuff ****/
void wObject::drawWireOutline(GLfloat xLen, GLfloat yLen,
                              GLfloat zLen, GLfloat centerX,
                              GLfloat centerY, GLfloat centerZ) {
    glColor4f(1,1,1,1.0f);
    drawWireRect(centerX, centerY, centerZ, xLen, yLen, zLen, 2);
}

/**** Axis Stuff ****/
void wObject::addAxis(GLfloat xLen, GLfloat yLen, GLfloat zLen){
    glPushMatrix();

    glPushMatrix();
```

```
    glTranslatef(0,yLen,0);
    glColor3f(0.0,1.0,0.0);
    gluCylinder(zAxis,0.1,0.25,zLen,30,30);
    glPopMatrix();

    glPushMatrix();
     glTranslatef(0,yLen,zLen);
    glRotatef(90,0.0,1.0,0.0);
    glColor3f(1.0,0.0,0.0);
    gluCylinder(xAxis,0.25,0.1,xLen,30,30);
    glPopMatrix();


    glPushMatrix();
    glTranslatef(0,yLen,zLen);
    glColor3f(0.0,0.0,1.0);
    glRotatef(90,1.0,0.0,0.0);
    gluCylinder(yAxis,0.25,0.1,yLen,30,30);
    glPopMatrix();


    glPopMatrix();
}


wObject::~wObject()
{
    //dtor
}
```

5.5 cluster_object.cpp - Everything to do with a cluster and a target

This file contains the class cluster_object, which is an object that contains everything that someone needs to know about the cluster loaded from the file. It also contains helper functions, for calculating the similarity between clusters and targets. cluster_object.cpp:

```
#include "cluster_object.h"
#include <iostream>
cluster_object::cluster_object ()
{
    return;
}
cluster_object::cluster_object (GLfloat l_x, GLfloat l_y, GLfloat l_z,
                                GLfloat c_x, GLfloat c_y, GLfloat c_z,
```

```
                                GLfloat s_weight, GLfloat s_type)
{
    len_x = l_x;
    len_y = l_y;
    len_z = l_z;
    center_x = c_x;
    center_y = c_y;
    center_z = c_z;
    weight = s_weight;
    type = s_type;
    axis = 'x';
    x_bound_low = center_x - (len_x / 2.0);
    x_bound_high = center_x + (len_x / 2.0);
    y_bound_low = center_y - (len_y / 2.0);
    y_bound_high = center_y + (len_y / 2.0);
    z_bound_low = center_z - (len_z / 2.0);
    z_bound_high = center_z + (len_z / 2.0);

    volume = len_x * len_y * len_z;
}
GLfloat cluster_object::get_similarity(std::vector<cluster_object*>* tgts,
                                       std::vector<cluster_object*>* objs)
{
    GLfloat ret_val = 0.0;
    GLfloat prec = 100; // precesion of similarity
    std::vector<point_set*>* obj_points; //points inside that object
    GLfloat percent_points = 0;
    GLfloat total_volume = 0;
    GLfloat total_target_volume = 0;

    //calculate the total volume inside the targets
    for (unsigned int u=0; u < objs->size(); u++)
    {
        total_target_volume += objs->at(u)->volume;
    }
    //calculate total volume so percent inside can be weighted properly
    for (unsigned int m=0; m < objs->size(); m++)
    {
        total_volume += objs->at(m)->volume;
    }
    std::cout << "Total cluster volume is: " << total_volume << "\n";
    //::NOTE:: will this method work right with overlapping volumes?
    for (unsigned int i=0; i < objs->size(); i++)
    {
        obj_points = objs->at(i)->get_interior_points(prec);
        GLfloat total_points_inside_me = 0;
        for (unsigned int j=0; j < obj_points->size(); j++)
```

```
        {
            GLfloat point_is_somewhere = 0;
            for (unsigned int g=0; g < tgts->size(); g++)
            {
                if (tgts->at(g)->is_point_inside_object(obj_points->at(j)->x,
                                                        obj_points->at(j)->y,
                                                        obj_points->at(j)->z))
                {
                    //denote that the point is somewhere in the targets
                    point_is_somewhere = 1;
                }
            }
            if (point_is_somewhere == 1)
            {
                total_points_inside_me ++;
            }
        }
        // caclulate the effect on the overall percentage by % of total volume
        percent_points = total_points_inside_me / (GLfloat)obj_points->size();
        ret_val += percent_points * ( objs->at(i)->volume / total_volume);
    }
    return ret_val;
}

bool cluster_object::is_point_inside_object(GLfloat p_x, GLfloat p_y, GLfloat p_z)
{

        if ( p_x <= x_bound_high && p_x >= x_bound_low
         && p_y <= y_bound_high && p_y >= y_bound_low
         && p_z <= z_bound_high && p_z >= z_bound_low )
    {
        // the point is inside this object
        return true;
    }


    return false;

}
std::vector<point_set*>* cluster_object::get_interior_points(GLfloat number_of_points)
{
    std::vector<point_set*>* ret_vector = new std::vector<point_set*>();
    GLfloat x_acc, y_acc, z_acc; // incremental distance between each point
    // along each axis

    x_acc = len_x / number_of_points;
    y_acc = len_y / number_of_points;
```

```
    z_acc = len_z / number_of_points;



    // 3 nested loops from bounds will get u num_points ^ 3
    for (int i=0; i < number_of_points; i++)
    {
        for (int j=0; j < number_of_points; j++)
        {
            for (int k=0; k < number_of_points; k++)
            {

                ret_vector->push_back(new point_set(x_bound_low + ( i * x_acc ),
                                      (y_bound_low + ( j * y_acc )),
                                      (z_bound_low + ( k * z_acc ))));
            }
        }
    }
    return ret_vector;

}

GLfloat cluster_object::get_percent_similiar(cluster_object* other_object)
{
    std::vector<cluster_object*>* vct = new std::vector<cluster_object*>();
    vct->push_back(other_object);
    std::vector<cluster_object*>* vct2 = new std::vector<cluster_object*>();
    vct2->push_back(this);
    return get_similarity(vct2,vct);



}
point_set::point_set(GLfloat xa, GLfloat ya, GLfloat za)
{
    x = xa;
    y = ya;
    z = za;
}
```

6 Results

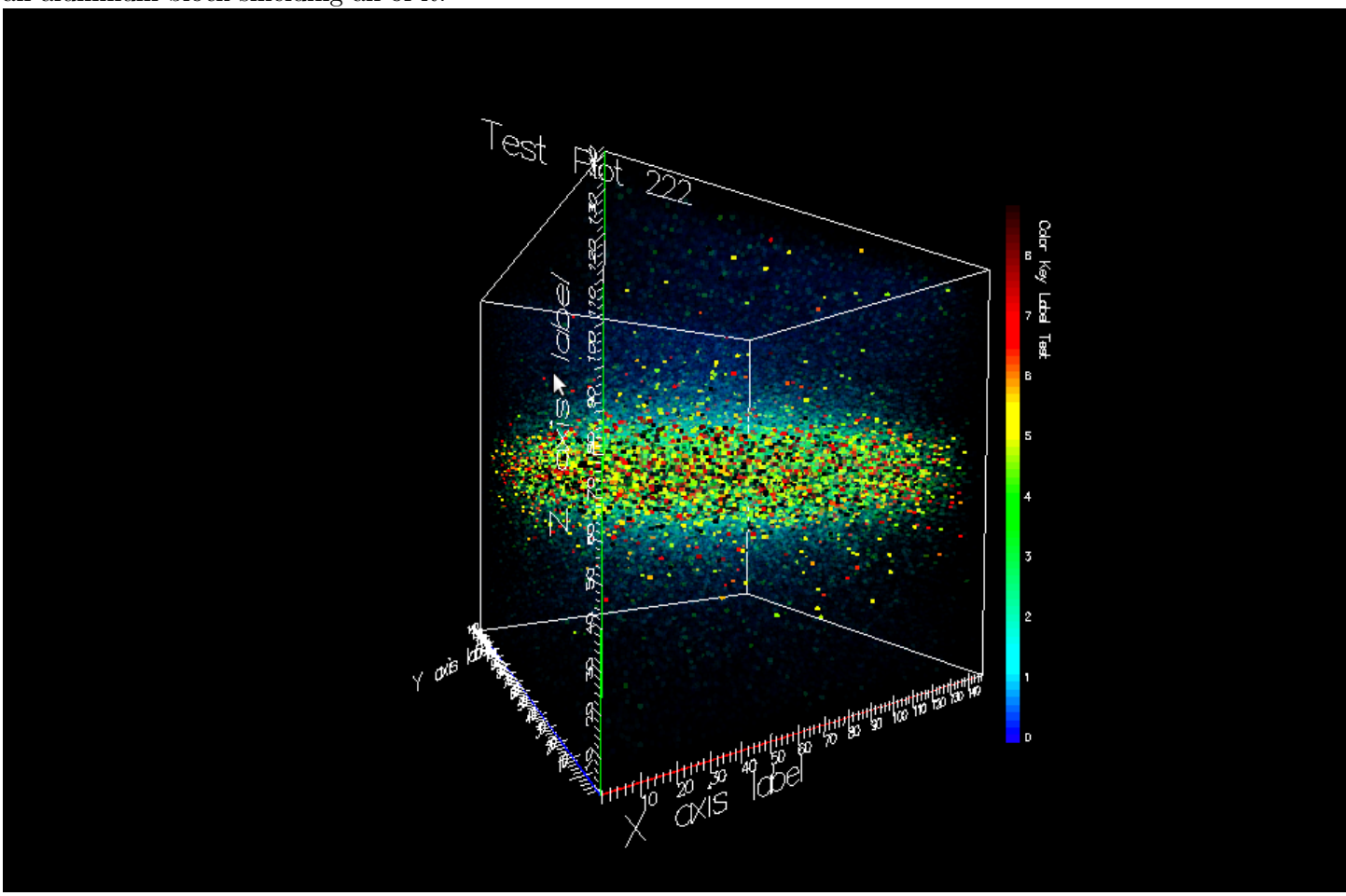
Due to the limiting time factor, a limited result set could be generated, since most of the time was spent coding. We have created results for the following sets:

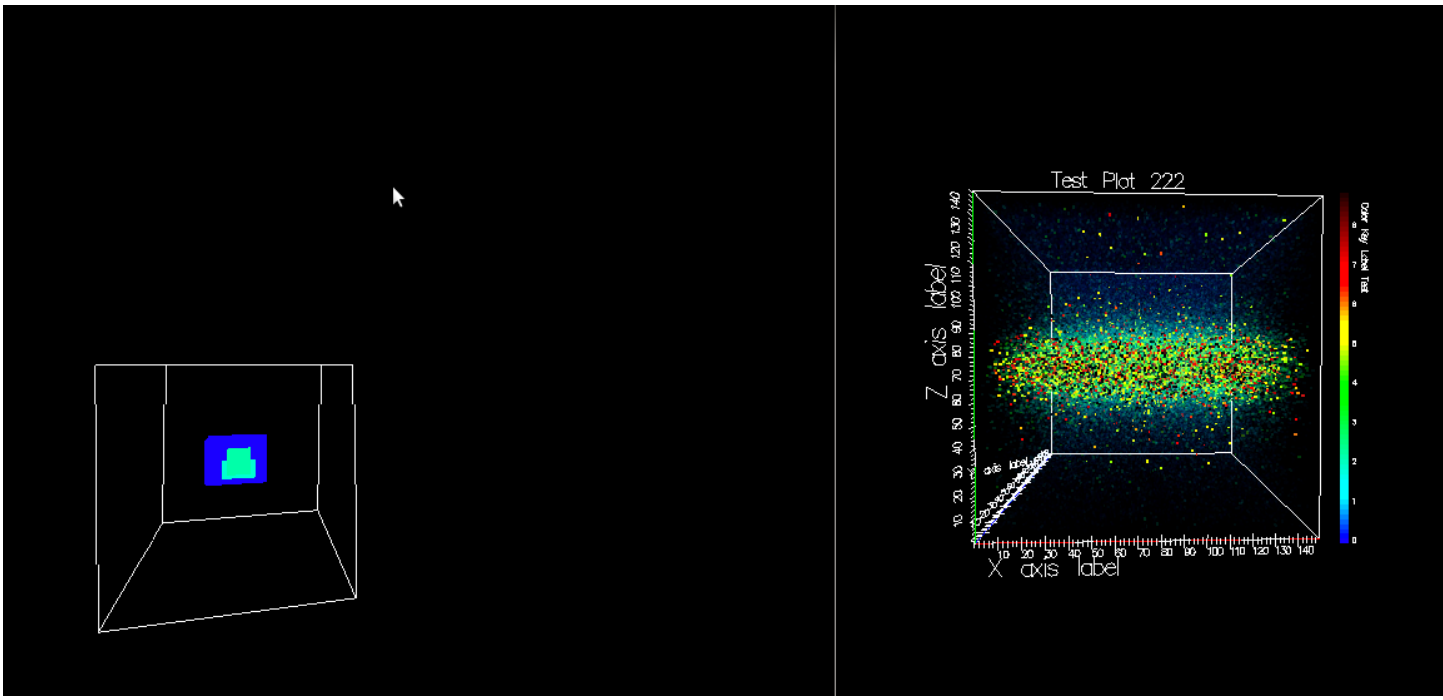
- fitShielded - High statistic monte carlo data of a shielded FIT uranium cutout in Al

- brassshield - Real data from the latest cubic foot mts station, of a brass shielded scenario
- stacked5Targets - stacked 5 targets data from Dec 1st from Cubic Foot MTS at Fl Tech

6.1 fitShielded

This is the Monte Carlo, ultra High Statistic setup, where there are F I T cutouts of uranium inside an aluminum block shielding all of it.





Output.txt:

```
cluster details# 1
mean angle: 3.968720
variance Angle:15.069571 deg2
variance (dX):0.000000 mm2
percent: 33.735868
Co-variance coefficient: -nan
#points: 3892.000000
centroid: -6.890622cm -0.531045cm 0.503926cm
lenght(X): 9.340953 width(Y): 9.326712 height(Z): 15.487751
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 296.392538


cluster details# 2
mean angle: 4.507397
variance Angle:15.999860 deg2
variance (dX):0.000000 mm2
percent: 40.628951
Co-variance coefficient: -nan
#points: 6328.000000
centroid: 0.643034cm 1.965876cm -0.267222cm
lenght(X): 11.314199 width(Y): 10.856211 height(Z): 8.218342
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 593.043733


cluster details# 3
mean angle: 3.782863
```

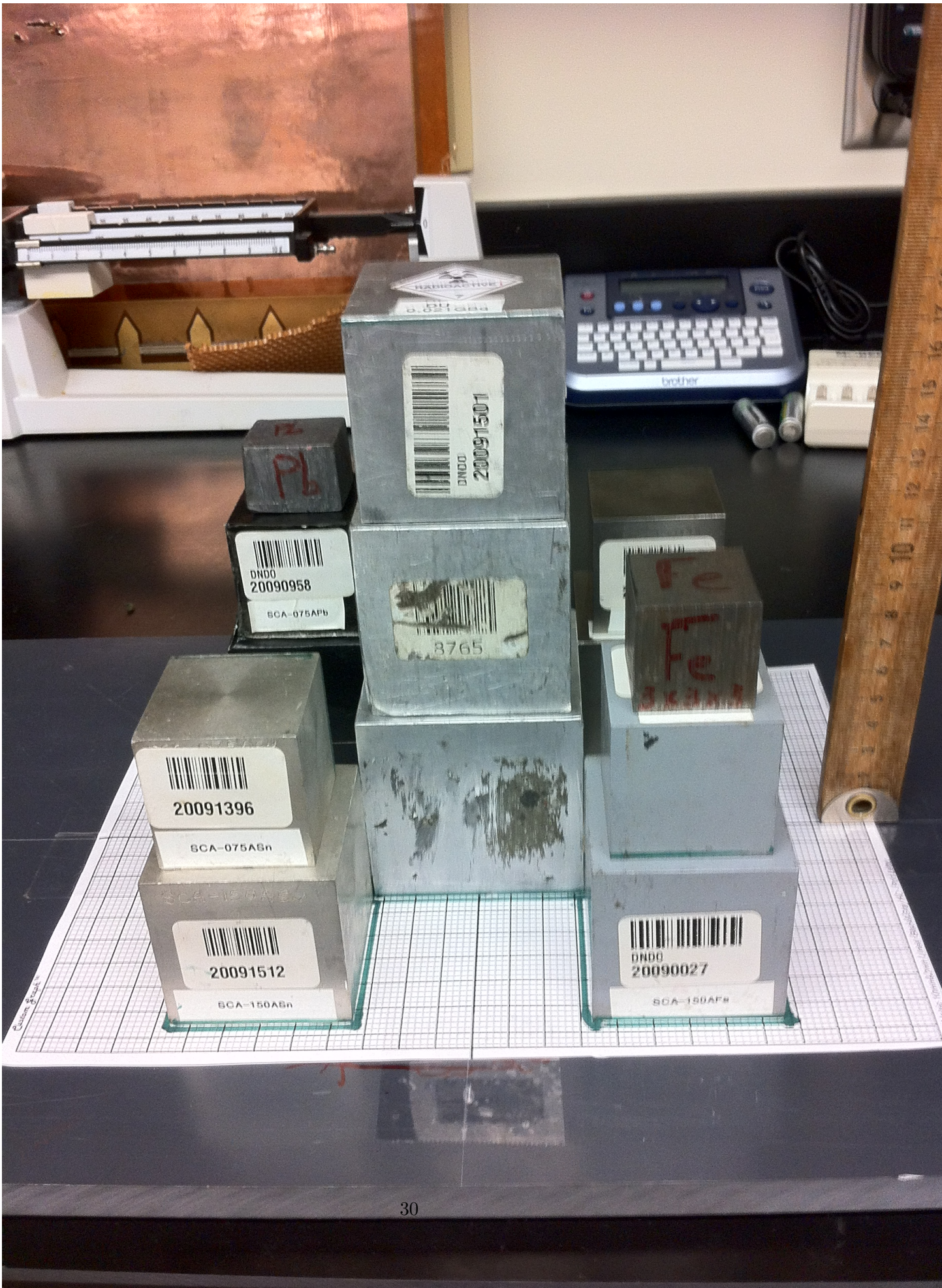
```
variance Angle:13.299155 deg2
variance (dX):0.000000 mm2
percent: 31.412866
Co-variance coefficient: -nan
#points: 4912.000000
centroid: 0.065958cm -6.456871cm -0.006745cm
lenght(X): 8.840672 width(Y): 8.920252 height(Z): 8.108295
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 499.630837


cluster details# 4
mean angle: 2.971149
variance Angle:9.573375 deg2
variance (dX):0.000000 mm2
percent: 22.605408
Co-variance coefficient: -nan
#points: 8728.000000
centroid: 4.243632cm 1.559873cm -0.857093cm
lenght(X): 38.391842 width(Y): 28.067735 height(Z): 19.026089
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 153.274623


cluster details# 5
mean angle: 3.342135
variance Angle:11.367796 deg2
variance (dX):0.000000 mm2
percent: 26.873602
Co-variance coefficient: -nan
#points: 3576.000000
centroid: -2.921180cm 9.310467cm 0.006179cm
lenght(X): 14.034476 width(Y): 6.705914 height(Z): 14.589169
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 237.356133


cluster details# 6
mean angle: 4.319032
variance Angle:19.239386 deg2
variance (dX):0.000000 mm2
percent: 38.861450
Co-variance coefficient: -nan
#points: 3847.000000
centroid: 0.783676cm -7.204300cm 0.798571cm
lenght(X): 20.812373 width(Y): 11.511311 height(Z): 23.046188
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 254.300282
```

```
cluster details# 7
mean angle: 5.693475
variance Angle:20.862331 deg2
variance (dX):0.000000 mm2
percent: 54.274240
Co-variance coefficient: -nan
#points: 5229.000000
centroid: 0.579649cm 3.179776cm -0.174710cm
lenght(X): 14.212110 width(Y): 16.485221 height(Z): 25.308557
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 251.102003


cluster details# 9
mean angle: 2.596880
variance Angle:7.548399 deg2
variance (dX):0.000000 mm2
percent: 17.756286
Co-variance coefficient: -nan
#points: 2585.000000
centroid: -7.793307cm -8.478862cm 0.399619cm
lenght(X): 8.451266 width(Y): 7.750033 height(Z): 16.012966
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 143.594495


cluster details# 12
mean angle: 1.646728
variance Angle:4.768154 deg2
variance (dX):0.000000 mm2
percent: 8.160920
Co-variance coefficient: -nan
#points: 870.000000
centroid: -8.015655cm 5.948019cm 5.552924cm
lenght(X): 11.010714 width(Y): 13.624884 height(Z): 13.515400
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 107.467249


cluster details# 14
mean angle: 4.131959
variance Angle:16.092036 deg2
variance (dX):0.000000 mm2
percent: 35.448005
Co-variance coefficient: -nan
#points: 3058.000000
centroid: 8.351283cm 0.633329cm 1.067887cm
lenght(X): 8.114715 width(Y): 14.168105 height(Z): 20.806082
```

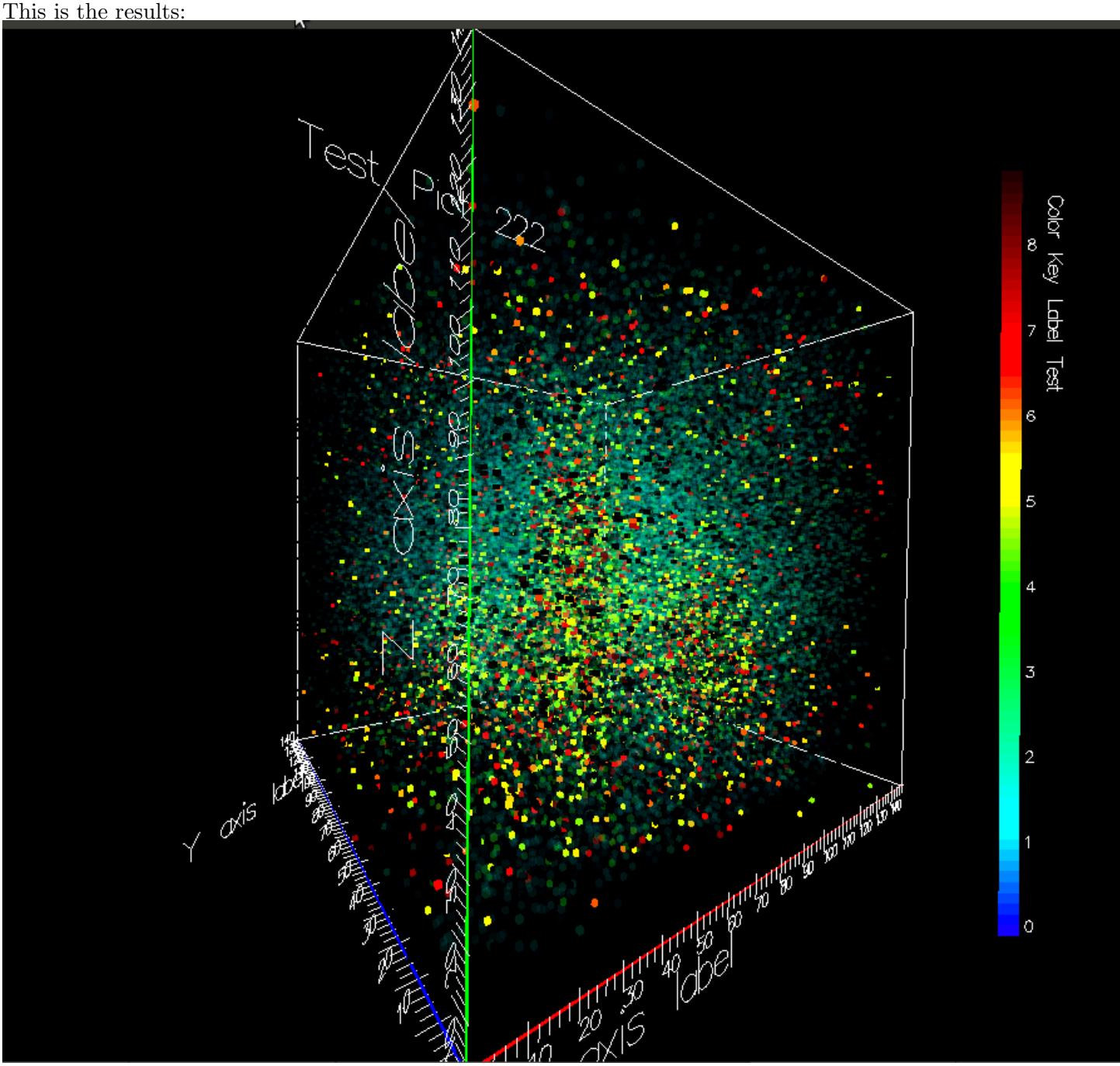
```
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 235.600072


cluster details# 15
mean angle: 1.712504
variance Angle:5.809783 deg2
variance (dX):0.000000 mm2
percent: 8.870968
Co-variance coefficient: -nan
#points: 744.000000
centroid: 5.820458cm 9.113126cm 5.436738cm
lenght(X): 12.490950 width(Y): 6.506743 height(Z): 13.318847
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 132.876452


cluster details# 16
mean angle: 2.013698
variance Angle:7.279368 deg2
variance (dX):0.000000 mm2
percent: 12.377451
Co-variance coefficient: -nan
#points: 816.000000
centroid: -7.388227cm -2.835841cm -5.514915cm
lenght(X): 11.744081 width(Y): 11.293135 height(Z): 6.653804
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 333.256344


File name: Ev10MTrgAlFePbWU40cm40cm20cm.txt
Total Time taken: 3.660000 seconds
#POCApoints: 210087
#Inside Box:135441
#Garbage added:47244
#Clusters: 17

enter fileNo:
ROC Analysis:
True Postives: 132689
True Negatives: 0
False Postives: 0
False Negatives: 2753
TPR: 0.979674
FPR:-nan
precision: 1.000000
accuracy: 0.979674
```

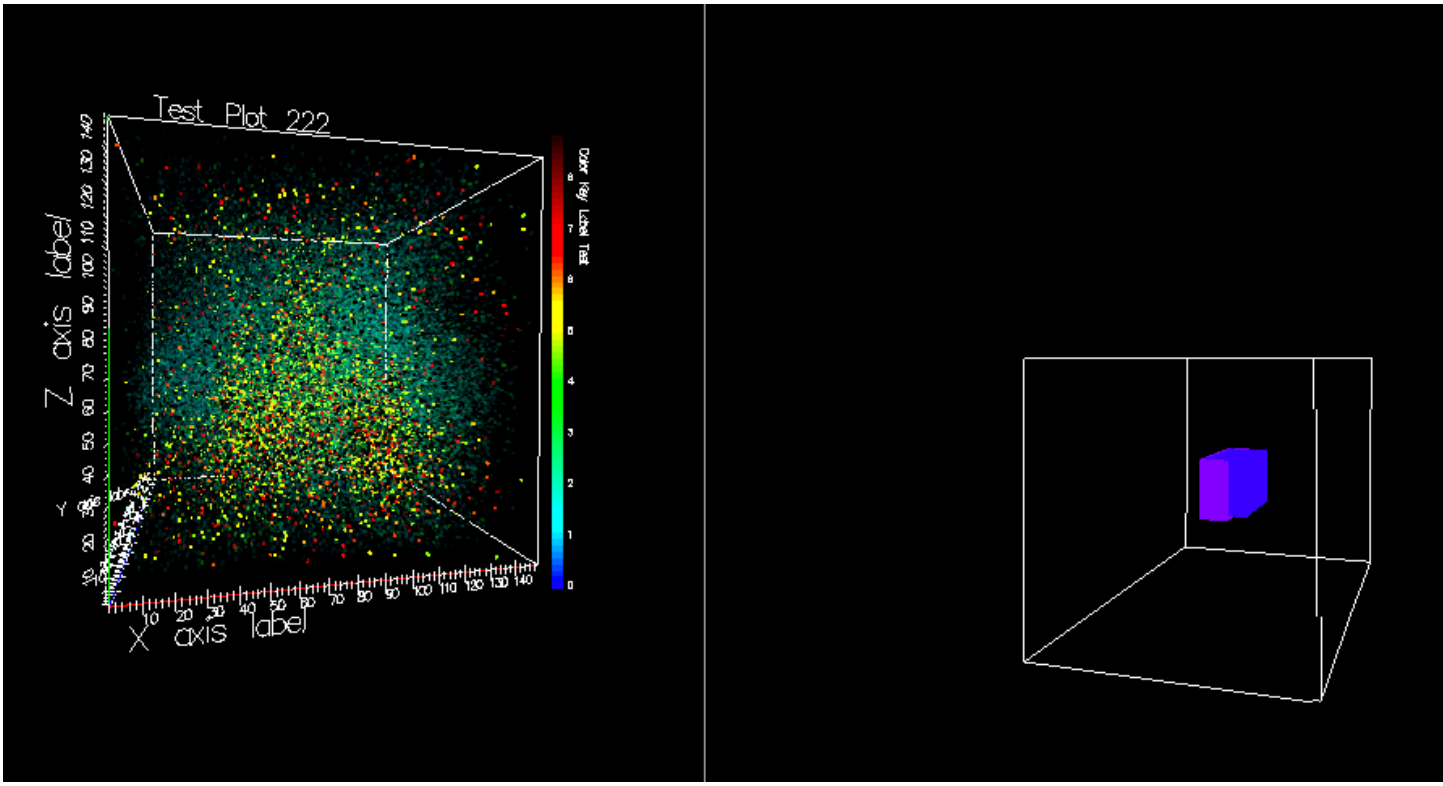
6.2 stacked5Targets

This was ALL of the stacked5 target files concatenated together, from the cubic foot MTS station
This is what the setup looked like originally:


DNDO
20091501
Pb
DNDO
20090958
SCA-075APb
8765
Fe
Fe
20091396
SCA-075ASn
20091512
SCA-150ASn
DNDO
20090027
SCA-150AFe
brother

This is the results:





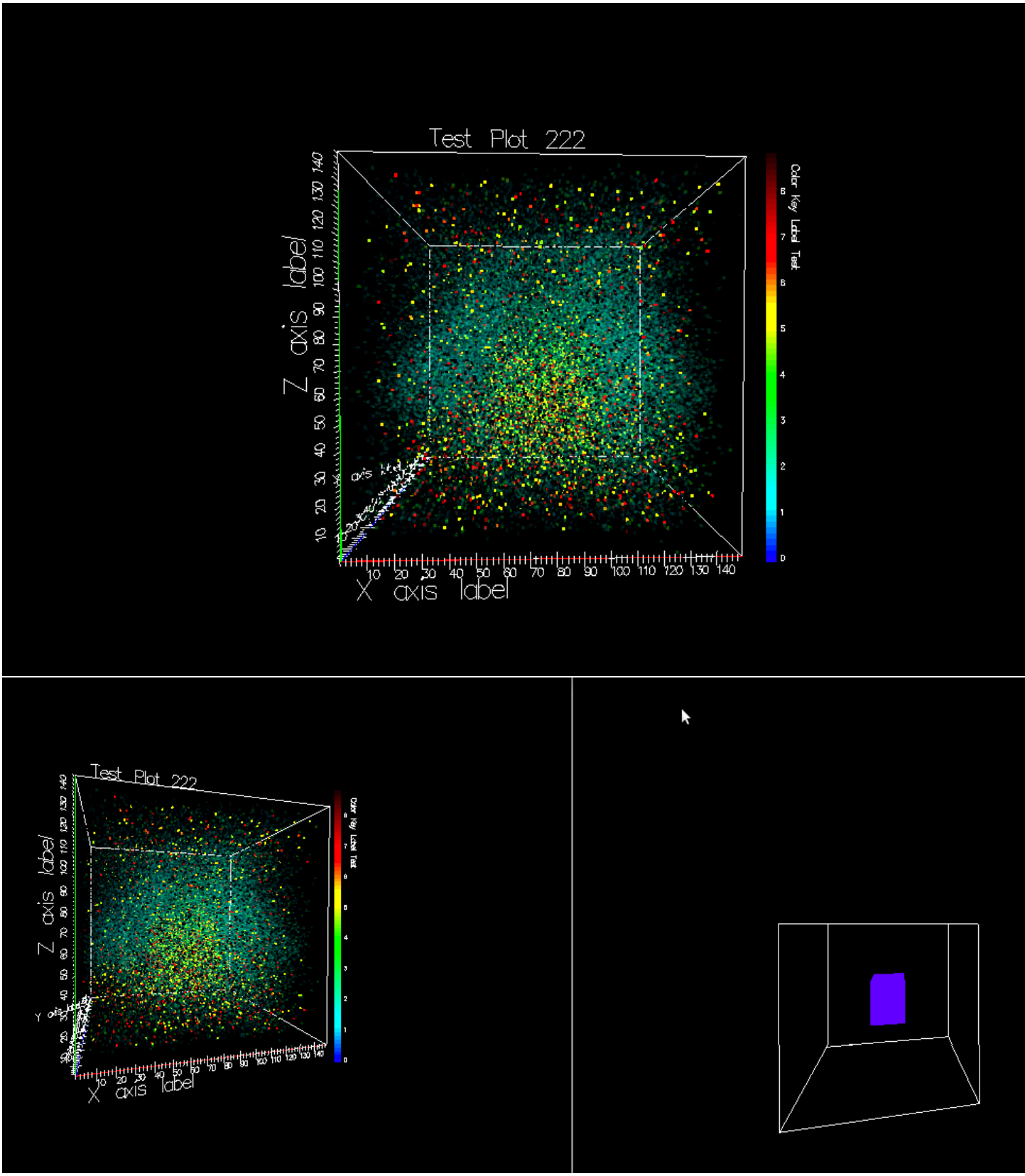
Output.txt:

```
cluster details# 1
mean angle: 2.245064
variance Angle:3.929168 deg2
variance (dX):0.000000 mm2
percent: 10.462287
Co-variance coefficient: -nan
#points: 2877.000000
centroid: -1.100818cm -1.539049cm 3.325185cm
lenght(X): 18.862214 width(Y): 17.523419 height(Z): 8.833820
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 135.489852


cluster details# 2
mean angle: 3.231034
variance Angle:7.930123 deg2
variance (dX):0.000000 mm2
percent: 24.761413
Co-variance coefficient: -nan
#points: 3877.000000
centroid: -0.200275cm -0.213953cm -4.047695cm
lenght(X): 16.661140 width(Y): 16.413363 height(Z): 12.742858
average P:3.000000 and p/p0: 1.000000
```

```
Estimated density at last: 189.569226


File name: Ev10MTrgAlFePbWU40cm40cm20cm.txt
Total Time taken: 0.290000 seconds
#POCApoints: 42642
#Inside Box:8634
#Garbage added:24829
#Clusters: 5

enter fileNo:
ROC Analysis:
True Postives: 8594
True Negatives: 0
False Postives: 0
False Negatives: 41
TPR: 0.995252
FPR:-nan
precision: 1.000000
accuracy: 0.995252
```

6.3 BrassShield

This data was taken from M. Staib's latest data collection of a secenario called "brassShield". It was all the files concatenated together, in g4hep's POCA output directory.



Output.txt:

```
cluster details# 1
mean angle: 2.355939
variance Angle:5.794199 deg2
variance (dX):0.000000 mm2
percent: 15.725151
Co-variance coefficient: -nan
#points: 123064.000000
centroid: 0.584253cm -0.120175cm -0.382009cm
lenght(X): 37.498795 width(Y): 34.436175 height(Z): 26.914406
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 65.578777


cluster details# 6
mean angle: 1.910141
variance Angle:4.172676 deg2
variance (dX):0.000000 mm2
percent: 10.517662
Co-variance coefficient: -nan
#points: 20071.000000
centroid: -3.472512cm -1.073550cm 2.777128cm
lenght(X): 22.788119 width(Y): 28.853709 height(Z): 22.072120
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 57.587119


cluster details# 9
mean angle: 1.338753
variance Angle:1.898727 deg2
variance (dX):0.000000 mm2
percent: 4.226804
Co-variance coefficient: -nan
#points: 5820.000000
centroid: 7.086928cm 2.448097cm 4.975493cm
lenght(X): 10.702174 width(Y): 19.809004 height(Z): 14.789675
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 39.107369


cluster details# 13
mean angle: 1.636068
variance Angle:3.260913 deg2
variance (dX):0.000000 mm2
percent: 7.477639
Co-variance coefficient: -nan
#points: 5590.000000
centroid: -7.577785cm 4.772446cm -4.424428cm
lenght(X): 9.272261 width(Y): 9.567239 height(Z): 8.454461
```

```
average P:3.000000 and p/p0: 1.000000
Estimated density at last: 117.491921


File name: Ev10MTrgAlFePbWU40cm40cm20cm.txt
Total Time taken: 31.830000 seconds
#POCApoints: 386272
#Inside Box:235135
#Garbage added:14579
#Clusters: 18

enter fileNo:
ROC Analysis:
True Postives: 235136
True Negatives: 0
False Postives: 0
False Negatives: 0
TPR: 1.000000
FPR:-nan
precision: 1.000000
accuracy: 1.000000
```

7 Plans for the future

- clean up the original algorithm code so it can be maintained and expanded in the future
- Look into ways into possibly lower the cluster density requirements for real data?
- Implement a parameter optimization algorithm that will work with parameters such as the ones in common.h
- Implement voxelized clustering instead of cuboid clustering