UNIVERSITA' DEGLI STUDI DI CAGLIARI

CORSO DI LAUREA IN INFORMATICA

3D RECONSTRUCTION OF VESSELS USING 'CGVIEW'

Advisor
Prof. Riccardo Scateni

Candidate
Hector Hernandez

ACADEMIC YEAR 2011-2012

Acknowledgements

I would like to thank Riccardo Scateni, my supervisor, for his support and dedication during this research and also to my colleagues of the Computer Graphics Group of the University of Cagliari. I also want to say thanks to my co-tutor Antonio Berlanga de Jesus, without him this wouldn't have been possible.

I also want to thank to Daniel Borrajo and Giorgio Giacinto, my academic coordinators in Spain and Italy, for helping me in everything I've needed all this year.

Thanks to all Erasmus and people of Cagliari that I met in this unforgettable experience.

Thanks to my family because without them I wouldn't be here and all the people I've met during my university life, that have changed my way of thinking and seeing things.

Finally, I don't want to forget about a very dear person, who will feel very proud in the top and he thought sometimes it would never end. All this work is for you.

Cagliari, September, 2012. Hector Hernandez

Contents

Illustration Table

1. Introduction

A large number of ceramic fragments, called “sherds”, are found at every archaeological excavation. These fragments are documented by being photographed, measured, and drawn. Archaeological finds are traditionally grouped by typology. Defined forms and types of vessels form codes which simplify communication within the scientific field. The drawing and interpretation of ceramic fragments is very time consuming and costly work, requiring trained and qualified draftsmen. The drawing in Illustration 1, for instance, is a representative of many other examples. The drawing is a 2D projection of the 3D object, therefore photographs of the real object have to be added. Nevertheless, there is no 3rd dimension left in the archive drawing and a graphic documentation done by hand also increases the possibility of errors. There may be errors in the measuring process (diameter or height may be inaccurate), and inconsistencies in the drawing of the fragment or the complete vessel. However, it is not possible to achieve a consistent style, since it is very difficult to make a drawing of an object without interpreting it. This leads to a lack of objectivity in the documentation of the material.

Because the conventional documentation methods were shown to be unsatisfactory, the interest in finding an automatic solution increased present a largely automated approach for estimating polynomial models in order to assemble virtual pots from 3D measurements of their fragments. Our approach to pottery reconstruction is based on the following main tasks: we start with the classification of the fragment based on its profile section provides a systematic view of the material found and allows us to decide to which class an object belongs. In the reconstruction phase, partial similarities of profiles can be detected and complete pots can be reconstructed based on the already stored data in the description At excavations most of the finds are in form of fragments, of which only a few are still complete. It would be ideal to have one acquisition system that covers both sorts of objects; however they have different properties (dimensions, color, and geometry).

Fragments of vessels are thin objects, therefore 3D data of the edges of fragments are not accurate. Furthermore this data cannot be acquired without placing and fixing the fragment manually, which is time consuming and therefore not practicable. Ideally, the fragment is placed in the measurement area, a range image is computed, the fragment is turned and again a range image is computed. This led us to the profile reconstruction method, which allows this kind of fast acquisition.

Traditional archaeological classification and reconstruction is based on the so-called profile of the object, which is the cross-section of the fragment in the direction of the rotational axis of symmetry. This two-dimensional plot holds all the information needed to perform archaeological research.

The correct profile and the correct axis of rotation are thus essential to reconstruct and classify archaeological ceramics. Illustration 1 shows the inner side of a fragment on the left, its left side (broken surface) in the middle, and the profile section generated automatically on the right. We follow the profile approach, as used by archaeologists for decades, to reconstruct complete vessels out of fragments.

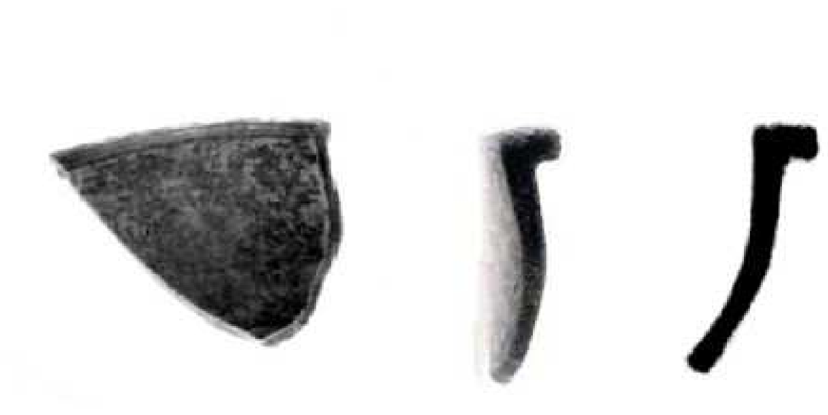

Illustration 1: (a) Archaeological fragment - (b) site of fracture and - (c) profile section.

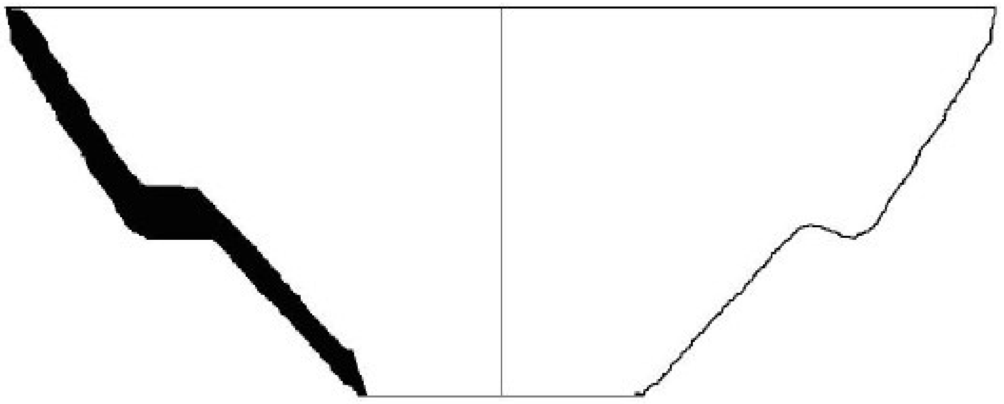

Illustration 2: Vessel's profile

2. Archaeological ceramics design

2.1 Introduction

The classification of ancient vessels of pottery is a fundamental part of the study of history.

Archaeologists around the world do this work using only hand tools like a caliber and a ruler. It is clearly understandable that the result is not perfect and the waste of time doing it, especially when it works with hundreds of pieces, is remarkable.

The result of this classification procedure is a schematic design of the vessel which remarks the most important characteristics of the same thing, (see Illustration 3).

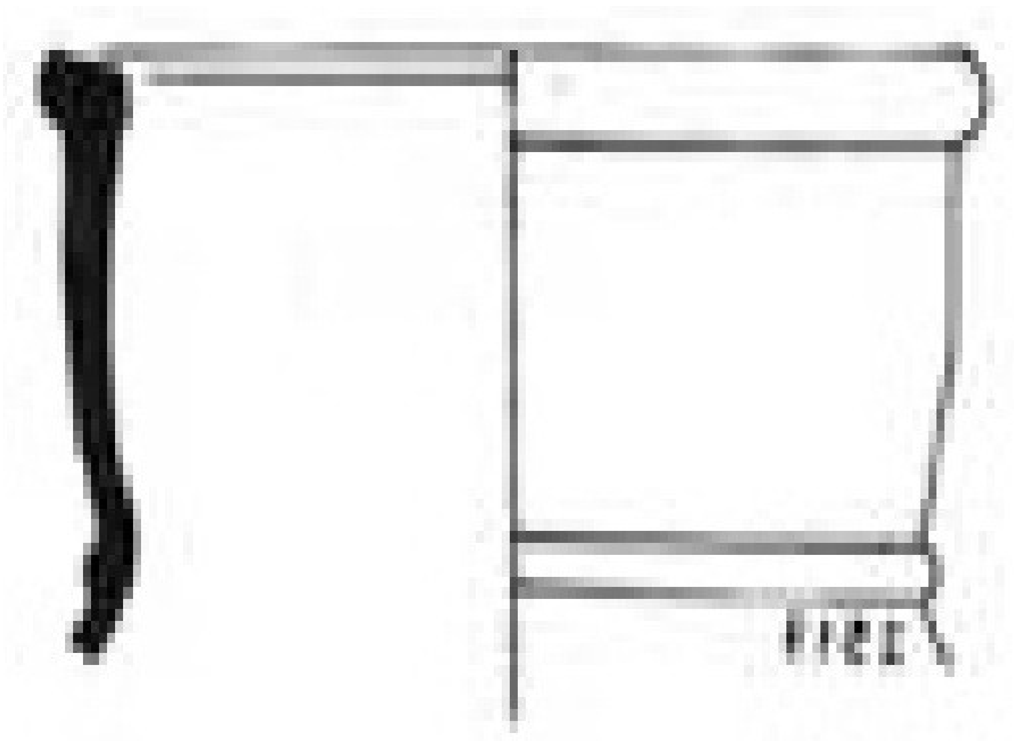

Illustration 3: Classification's example

The design complements the cataloguing of materials, it is useful to the typology's articulation, accompanies and explains in the text's edition phase, to replace it with whatever concerns the morphological description.

In every case, an object's drawing must be comparable with other created within the same research, but also outside, and must be made according to unique criteria that would make its understanding universal.

This requirement meets the articulation of a set of rules meant to ease the design, freeing it from heavy esthetics characterizations (the real picture the object is provided to best right from a photograph) and to give prominence to morphological information.

The designer makes mediation and chooses to represent some details rather than others (delete certainly any "recent" traces derived from the deposition or withdrawal of the object; the photography however doesn't document the state without any type of interpolation).

So, it's essential an understanding of the materials types and manufacturing techniques in order to understand the object before drawing it.

The current trend is to free the drawing surface by redundant information (for example the conjunction lines of fragments) to emphasize the overall grade of vessel's integrity, the moldings and the lines resulting from shaping (lathe lines, tracks cue, fingerprints etc.).

2.2 Design settings

The ceramic's design involves transposing a three-dimensional object into a plane, through an orthogonal projection and by the paper's organization in areas intended to clarify information about the different parts of a pot.

There will be primary information, as basic and indispensable, intended to exemplify the form (one or more sections of the ceramic body) and size (diameter of the edge and/or bottom, height), and secondary information — because it depends on the first and to add more data — documenting the perspective view, decoration or other interventions which are on the vessel's surface.

Switching to 2D drawing involves setting one (fragments) or two (in the case of artifacts that preserve all the profile) horizon lines (horizontal and then parallel if the pot is not deformed) and an orthogonal line, median, which divides the space-paper in two areas: the left one is for the design of the vessel section and the eventual information on the intern surface, the right, however, is intended to outline's perspective view of the external surface.

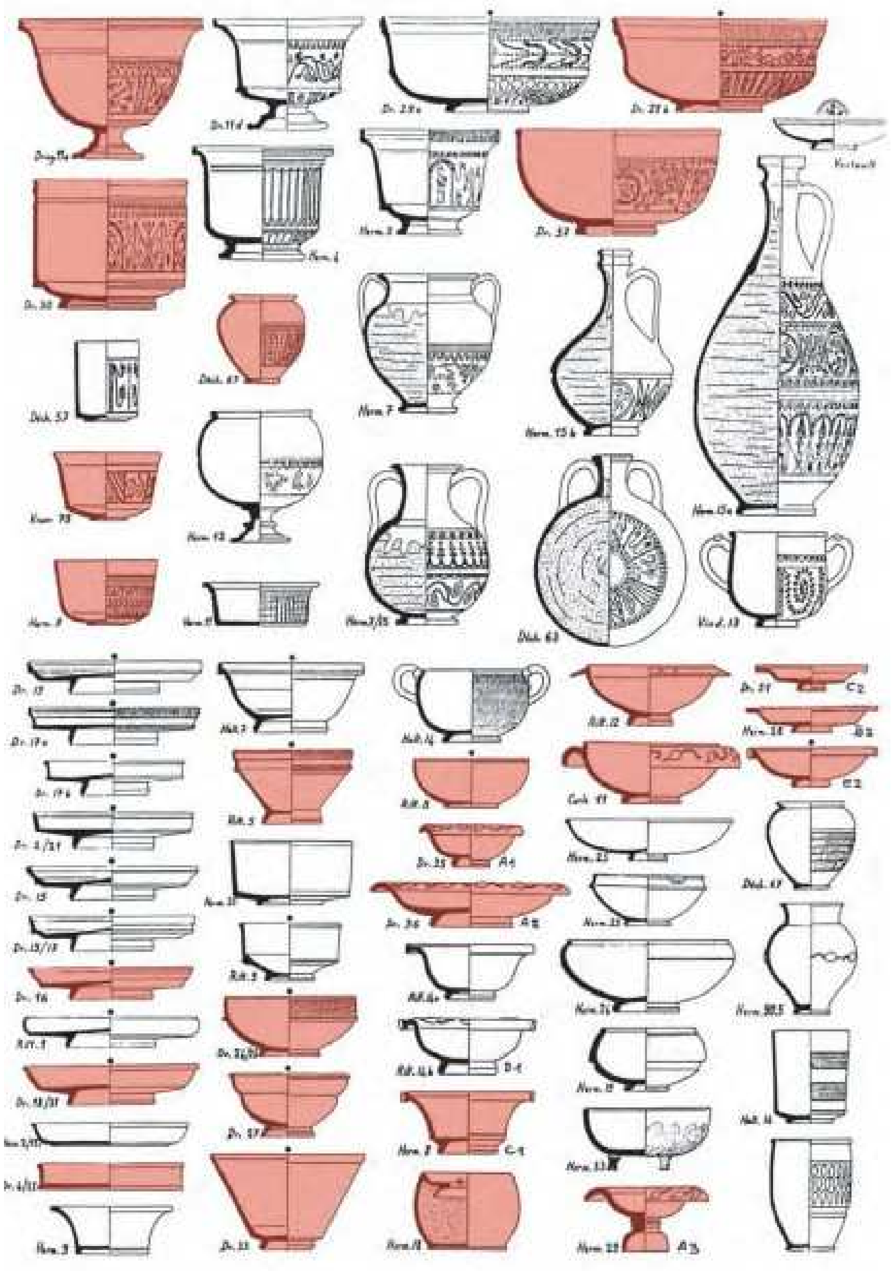
Illustration 4: Set design

The distance between the two horizon lines is given by the pot’s height.

The above and below spaces of the horizon of upper and lower lines can be used for further information on special particular vessel’s conformations or decorations and outer bottom’s surface, (see Illustration 5).

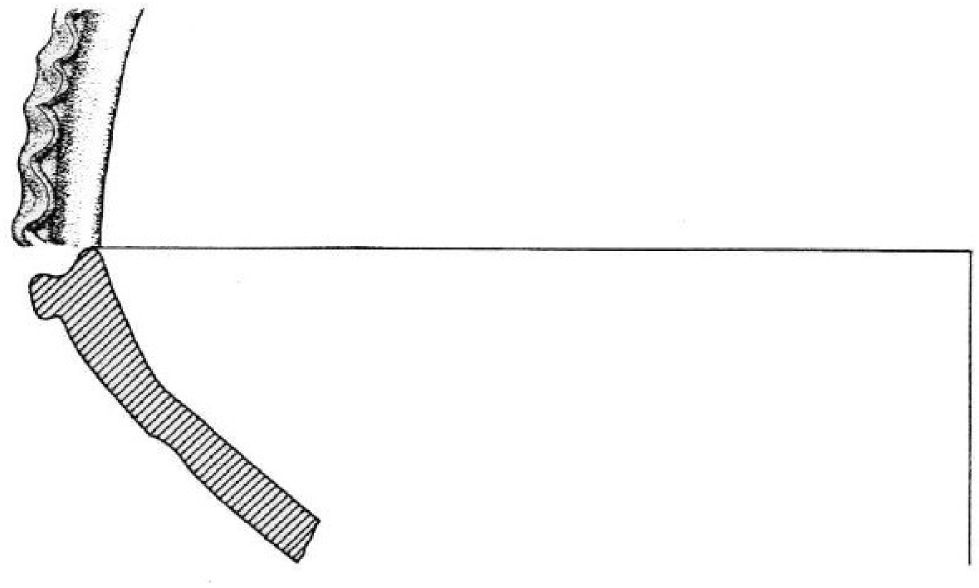

Illustration 5: Vessel's decoration

Horizon lines have the function to determine the object's inclination. The design must be oriented respect to a horizontal plane, must document the profile in a point of maximum conservation, but can't be in any reconstructing way.

2.3 Whole vessel's designs

The design starts from the section. These facts, invisible especially in the presence of integral vessels, derive from the sum of the exterior and interior profiles, detected along a vertical axis which ideally cuts the ceramic body. In the vessel's case, or preserving intact the whole profile, drawing was uses three fundamental data: the external measure of upper and lower diameters, and of the height. The profile's inclination is given by the difference between the two diameters. The drawing normally proceeds from bottom to top: It's set the lower horizon line, then the height and the upper horizon line (parallel to the bottom if the vessel doesn't have deformations).

Diameter measurements are easily detectable with a ruler in the whole vessel or that conserve the edge for a superior portion to the middle circumference. The height is also measured with an squadron or with a ruler, counting on some precautions: in paper's transposition on detected measure (that it's obtained supporting the carpenter's square on the table vertically, tangent to the point of maximum vessel expansion) we must count on the space more or less wide than the instrument presented before the start of measure marks; the height must be detected in correspondence with the superior border and not of the external border; squadron must be placed with 90° a ngles, (see Illustration 6).

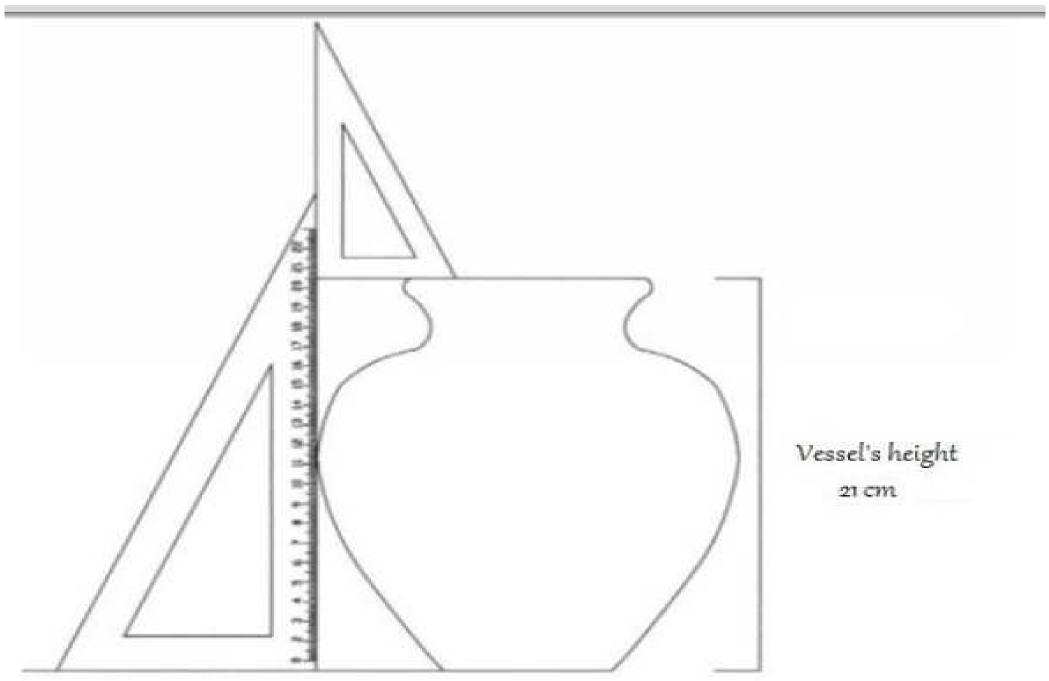

Illustration 6: Squadron's position

If the circle is preserved for a lower portion in the middle is necessary to rebuild the diameter's measure. This operation can be done using a "Rim Diameter Chart" (paper prepared with a series of concentric circles in a distance of 5 or 10 mm), (see Illustration 7).

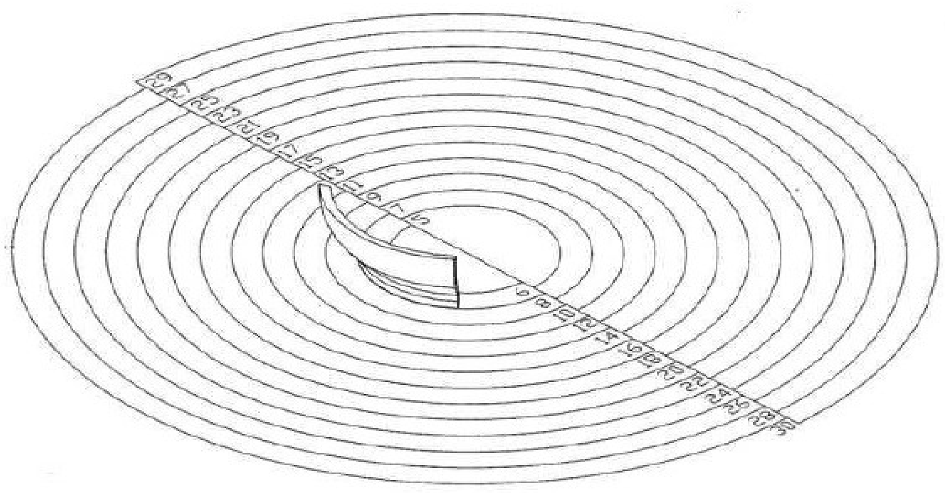

Illustration 7: 'Rim Diameter Chart'

Or using a geometric application which allows to rebuild the axes at least two strings to the circumference arc (Theorem: in a circle, the axis of a string passes through the Centre), axis that determine, at the intersection point, the center.

Then it’s reported on the paper the external or internal circumference’s arch conserved (making to join the vessel’s edge to the paper by using a “profilometer”), (see Illustration 8).

Illustration 8: ‘Profilometer’

Are identified on the designed arc at least three well-preserved points (three unaligned points are passed by one circumference) and trusted then (A, B, C) and with a compass tracing its circumferences (the arch opening of the compass is indifferent); we draw the lines that pass in the circle’s intersection points (corresponding to axes of strings AB, BC) and the meeting point determines the center, (see Illustration 9).

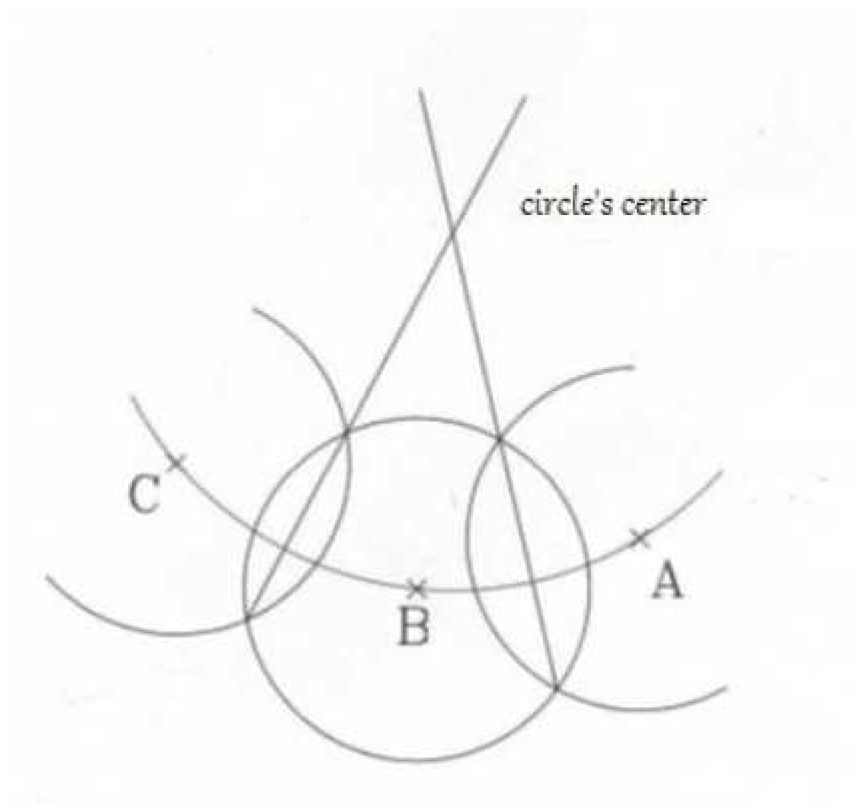

Illustration 9: Center's calculation

In the presence of a small pot, the three basic measures are sufficient to complete the drawing of the profile: this operation is essential to the use of a profilometer, that allow to detect with extreme precision the imprint of the object (in order not to ruin the surface of the vase is good to push towards the needles the piece, rather than the opposite).

In the presence of vessel larger than the length of the profilometers you must detect other points on the external surface (more or less corresponding to the highlights: maximum expansion of the abdomen, moldings). As for the measurement of height, this operation is carried out with a team placed in the vertical position, tangent to the pot: you can obtain the coordinates (x, y) to report on paper, taking into account the arrangements already exposed.

The design of internal profile is run with the profilometer, having noted with a caliber and in several places the thickness of the wall and the base. In the presence of "closed" pot it's possible that the profilometer is not usable, if not perhaps for a short stretch corresponding to the mouth: the drawing of the profile inside can be accomplished through a thick pad thickness variation ceramic body, (see Illustration 10).

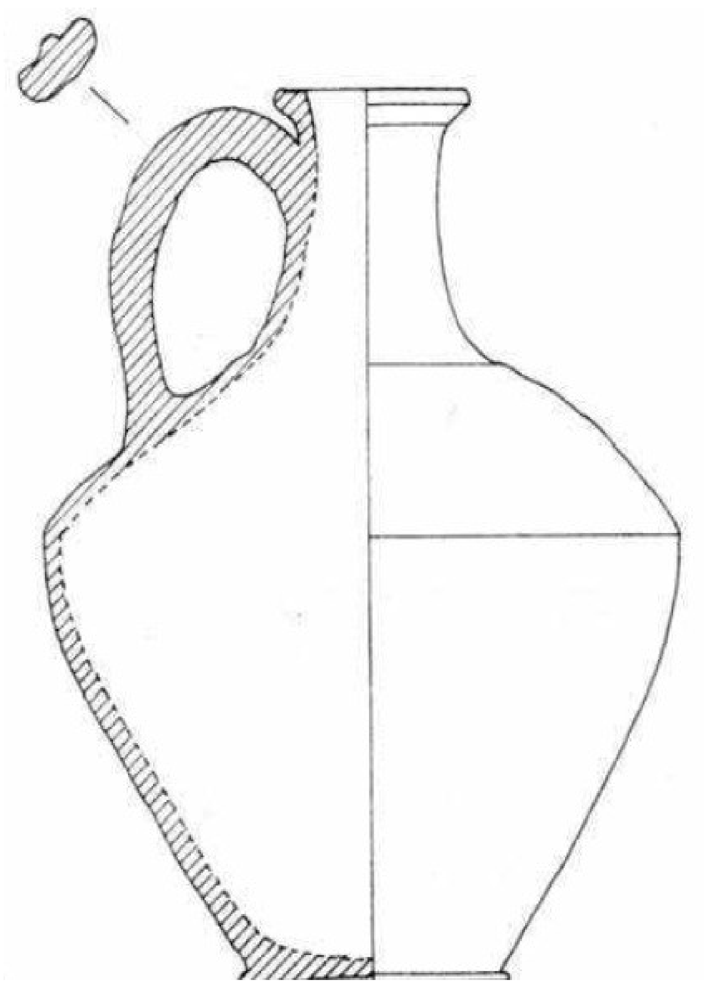
Illustration 10: Amphora

2.4 Fragments design

A fragment is drawn if it contains useful information for a morphological or decorative repertoire classification. Must then submit an edge or a base retained enough for the purposes of determining the orientation.

The design of a wall fragment (designed for the presence of a decoration or a handle of type note) can be done by using, for the determination of its inclination in space, the lines of lathe, the hull or any molding, elements that, if enough preserved, also allow the reconstruction of the diameter.

The design of a fragment sets up a single horizon line, corresponding to the edge or to the bottom and proceeds from bottom to top or from top to bottom as a result of the preserved portion.

For the design of a bottom the procedure will be very similar to that adopted for a whole vessel: the diameter measure will be rebuilt and the given diameter will be reported on the paper corresponding to an horizon line in the lower part of the paper; the vertical medium line will be drawn corresponding to the object's axis for a height equivalent to the degree of the fragment conservation.

The inclination will be calculated supporting the bottom on the paper, so its outside diameter matches the designed point in a way that the supporting surface is kept adhered to the table. With a squadron the measures corresponding to the maximum conservation point can be known (coordinates).

The external profile can be now done with the help of profilometer. The thickness can be registered easily also with a simple scrolling caliber.

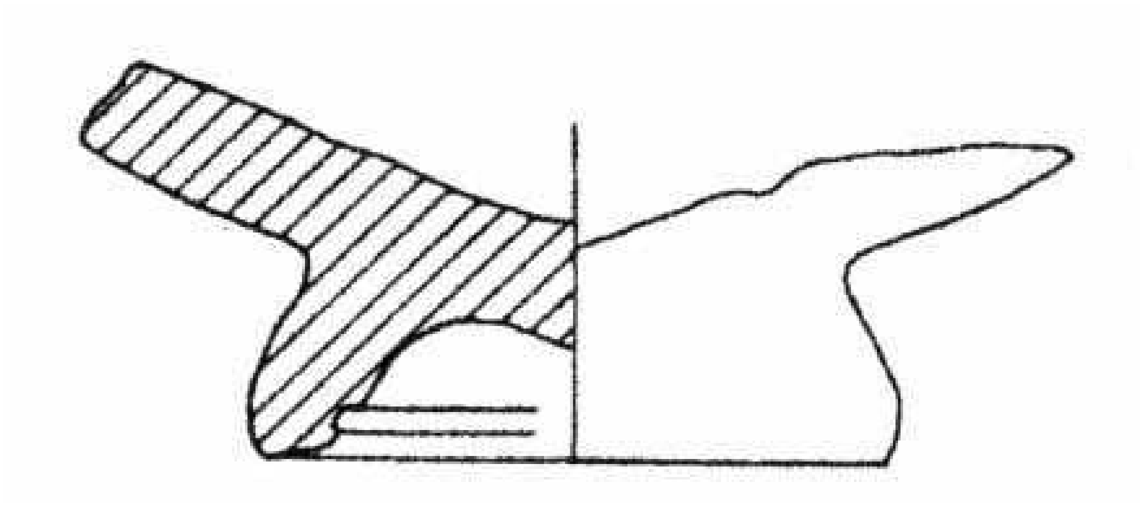

Illustration 11: Wall fragment

For the design of an edge it will be necessary to overturn the procedure: the measures will be taken from the top. Then it will be drawn an horizon line in the upper part of the paper and the midline will be developed with a height corresponding to the maximum conservation of the fragment. The first useful data is, however, the reconstruction of the diameter measure that will be given on the paper on the horizon line.

The following measures (coordinates of one or more points) and the fragment's inclination will be detected by supporting the edge on the paper, so its outer diameter matches the point designed in a way that the supporting surface is adhered to the table.

With a squadron can be detected the measures (coordinates) corresponding to the maximum point of storage required on the paper.

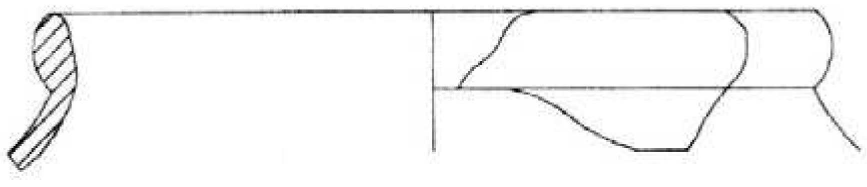

Illustration 12: Broken vessel

Fragments of a diameter not determinable. The degree of conservation impact not only on possibility of identifying the size of a pot (with a possible error identification within the same functional form anyway), but also on the determination of its inclination and its functional form (a pot becomes a cup a dish) or, at best, its typology.

The graphic criteria for representing fragments which, while allowing the recognition of an inclination, do not allow the reconstruction of the diameter, provides the suppression of median line, (see Illustration 13).

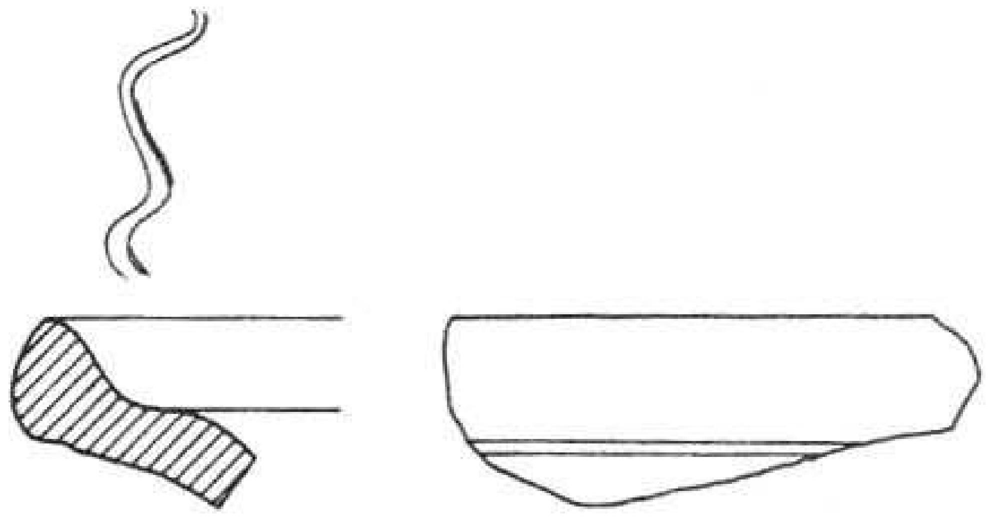

Illustration 13: 'Sherd' design

3. CGView

3.1 Introduction

As the interest in Computer Graphics increased at the University of Cagliari, the need for tools that allowed to experiment and study the topic became stronger. While such tools, as MeshLab or OpenFlipper, are free, versatile and offer a great number of possibilities for the development of mesh processing algorithms, their complexity makes them quite unfriendly to the newcomer, not allowing to easily lay hands on every aspect the term Computer Graphics covers; moreover, teaching can benefit from tools that allow a direct experimental approach, where the student has to deal with simple codes with everything in sight, with no interfaces or library calls that can obscure the whole functioning, thus allowing to endure a trial and error approach when modifying the program code to see if the results agree with what expected.

Finally, the recent research activities carried on by the group focused on aspects that went outside the scope of the mere Mesh Processing; these reasons brought to the development of a simple and versatile family of applications where each student or researcher can focus directly on the goal of his work, from rendering to high-level data visualization, without having to deal with the complex infrastructures that advanced tools as the ones cited before are made of.

The need to coordinate the works of many people and help students to get acquainted with Computer Graphics problems has brought to the development of light, versatile and easily usable tools for visualization and processing. Gathered under the name of CGView, these projects form a continuously work-in-progress framework that helps the programmer to easily focus on each aspect of the problem, from accessing GPU buffers to the visualization of data. The CGView family is focused on two main goals: giving the student the possibility to view and explore the 3D environment with direct understanding of the code involved, and allowing a programmer to develop a mesh processing algorithm without having to deal with complex interfaces.

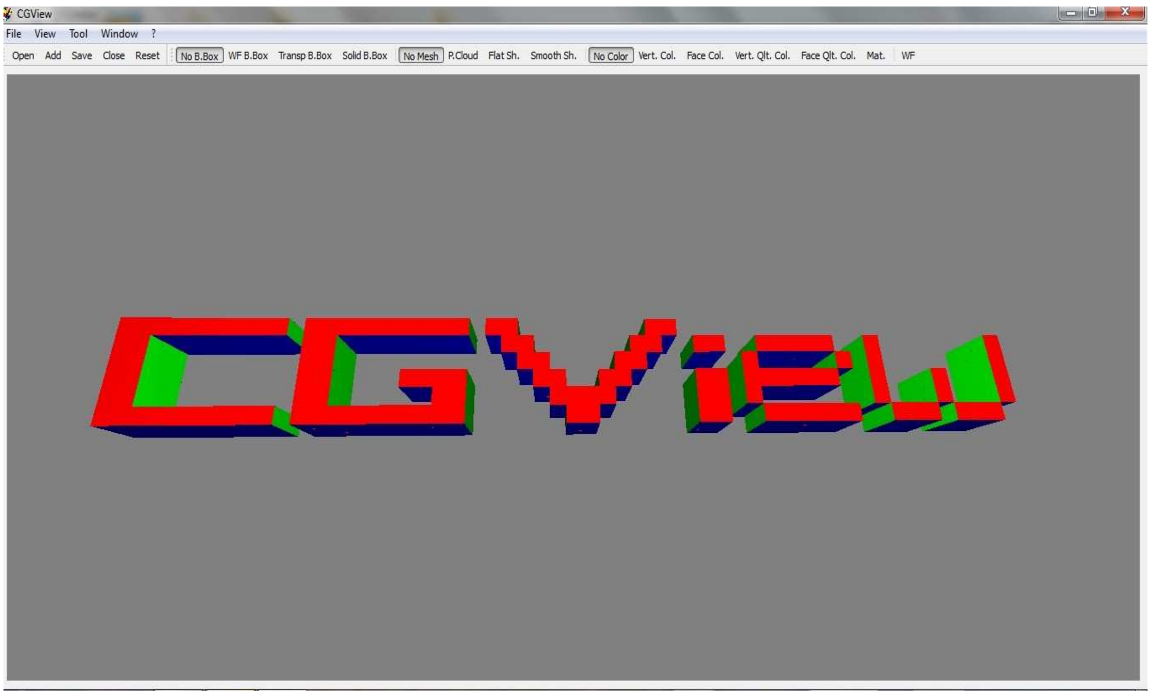
CGView
File View Tool Window ?
Open Add Save Close Reset
No B.Box WF B.Box Transp B.Box Solid B.Box
No Mesh P.Cloud Flat Sh. Smooth Sh.
No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat.
WF

Illustration 13: CGView

Based on the VCG library for mesh processing and the Qt Toolkit for its interface, CGView is capable of opening and managing other kinds of data such as voxel grids or topological skeletons.

The application can visualize a mesh in every usual presentation as wireframe, points, flat or smooth shaded and can add materials into rendering. Moreover it can visualize all the main information about the mesh, such as bounding box, axes, normals, etcetera. However, as said before, the main goal of the application is to give the student the possibility to experiment and easily add its own work into the program, so the features are in constant upgrade.

As for mesh processing, the application takes advantage of the methods provided by the VCG library along with some added algorithms developed by researchers and students using one of the satellite applications.

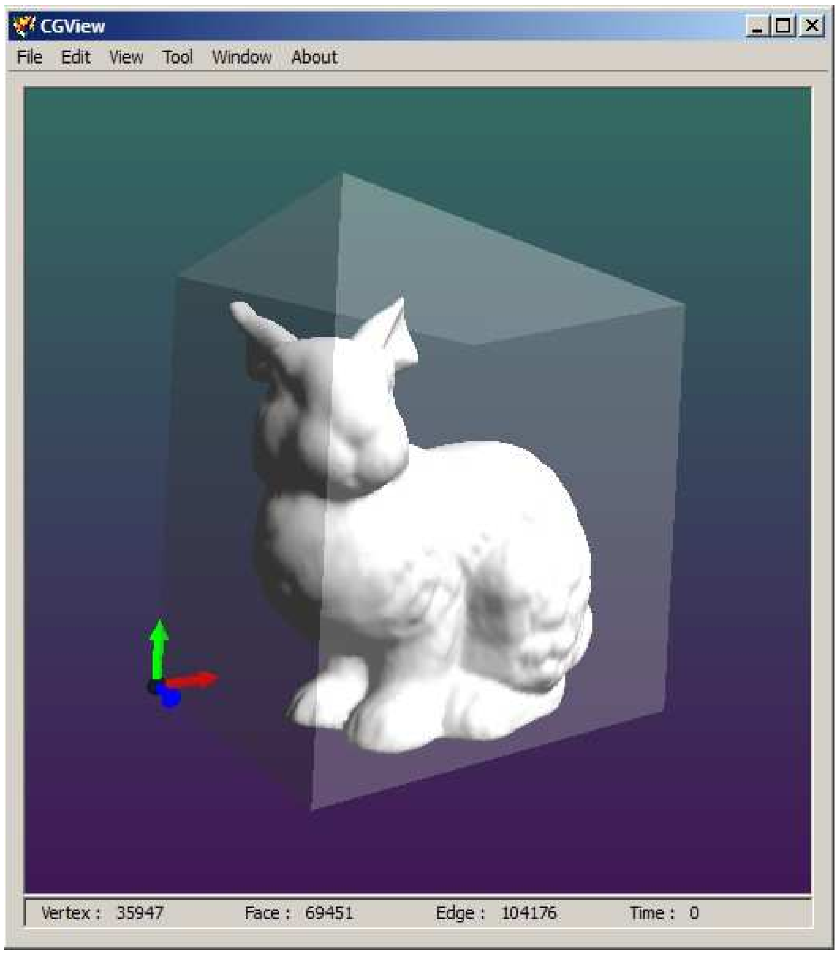

Illustration 14: The main window of CGView showing the mesh, its bounding box and the axes

3.2 Objects and functions

3.2.1 Point

A point is a set of 3 values (x, y, z) where these values indicate a point or a vector in the Cartesian space.
A point is an object that is declared as follows:

CGPoint < name > point;

- **P [i]** returns the i-th value of item with which goes from 0 to 2. Values are double.
- **P. X ()** return the point x. Is equivalent to P [0].
- **P. Y ()** return the point's y. Is equivalent to P [1].
- **P. Z ()** return the point's z. Is equivalent to P [2].
- **P. Norm ()** returns the norm of P.
- **P. Normalize();** normalizes P.

Some operations are also defined on these objects. If **A** and **B** are objects of type CGPoint and **k** is a scalar, then:

- **P = A + B;** P is the vector sum of A and B. P = (Ax + Ay + Bx, By, Az + Bz). The same effect is obtained by P [0] = A [0] + B [0]; P[1] = ...
- **P =-P;** Reverses P. = (Px, Py, Pz--).
- **P = A * k;** P is the result of the scalar product between A and k. P = (Ax, Ay * k * k, Az * k).
- **P = A ^ B;** P is the result of the wedge product of A and b. P is orthogonal to vector A and b.
- **A = B =** returns true if the two points are equal.

There are also functions that work with points:

- **vcg:: Distance (CGPoint, CGPoint b)** that returns the distance between 2 points (double).
- **vcg:: Angle (CGPoint, CGPoint b)** that returns the angle between 2 vectors (double).

If you need to rotate a vector will define the rotation matrix. To do this you have to (if not present between objects in the namespace vcg) include the 3 x 3 matrix:
# include <vcg/math/matrix33.h>

Now you can declare an object **vcg:: Matrix33 <double>** (or **vcg:: Matrix33d**). This object is like a 3 x 3 array, so you can look for an item by using the indexes:

```
…
vcg:: Matrix33d rot;
…
rot [0] [0] = ...//element 0 row and 0 column
rot [1] [2] = …//element 1 and line 2 column
…
```

For a rotation:
rot. SetRotateDeg (*<valore>, axis*)
Depending on whether you want a rotation in degrees or radians, generic around *axis*vector. Once set the matrix you can use **P = rot * A;** that assign to P the result of A rotation.

3.2.2 Vertex

The vertex is an evolution of type CGPoint. Is defined **CGVertex** type and consists of some additional features compared to CGPoint. Firstly a vertex has both a position in space that a normal, then a color and a whole host of more or less useful flag. Is declared like this:

CGVertex < name > Vertex

- **V. P ()** return a CGPoint, indicating its position in space
- **V. N ()** return a CGPoint, indicating his normal

- **V.C ()** return the color as object vcg:: Color4b (). Is a vector containing the values RGBA color.
- **V. SetS ()** set the vertex selected.
- **V. ClearS ()** set the vertex as unselected.
- **V. IsS ()** returns true if the vertex is selected.
- **V. SetV ()** set the vertex as visited.
- **V. ClearV ()** set the vertex as unvisited.
- **V. (IsV)** returns true if the vertex is visited.
- **V. P () = A;** assigns as the position at the vertex V
- **V. N () = A;** assigns as normal vertex V
- **V. C () = C;** assigns the color C at the vertex V.

3.2.3 Face

The faces are nothing more than a collection of 3 vertices, one normal, one color and of various flags. A face is defined as type CGFace. If **F** is a CGFace, then:

- **F. P (i)** return the coordinates of the i-th vertex (CGPoint) ranging from 0 to 2.
- **F. V (i)** return the i-th vertex (a CGVertex), which goes from 0 to 2.
- **F. N ()** return the normal of the face (CGPoint).
- **F. C ()** return the color of the face (Vertex).
- **F. FFp (i)** returns a pointer to the i-th face adjacent to F.
- **F. SetS ()** sect face as selected.
- **F. ClearS ()** sect face as unselected.
- **F. IsS ()** returns true if the face is selected.
- **F. SetV ()** sect face as visited.
- **F. ClearV ()** sect face as unvisited.
- **F. (IsV)** returns true if the face is visited.

3.2.4 Mesh

A mesh is a collection of vertices, edge and triangles. In VCG has no concept of edge in the strict sense of the term (there is an object of type Edge), for which mesh are formed only from lists of triangles and vertices. A mesh is defined as:

CGMesh < name > mesh;
Essential attributes that need to know this kind of object are few.
If for example we **M** object of type CGMesh, then:

- **M.vn** return the number of vertices in the mesh. Is an integer.
- **M. vert [i]** return the i-th vertex. The vertex is an object type CGVertex.
- **M. fn** return the number of faces in the mesh. Is an integer.
- **M. face [i]** return the i-th face. The face is an object type CGFace.
- **M. bbox** return the bounding box of the mesh in the form of vcg:: Box3 object.

The vertices and faces in the mesh should be organized on the vector (i.e. std::vector), M.vert.begin () returns the iterator to the first position of the vector, while M.vert.end () at the end. The same things apply to faces

Through the CGMesh class you can then declare some data types, including:

- **CGMesh:: VertexPointer** pointer to vertex.
- **CGMesh:: FacePointer** pointer to the face.
- **CGMesh:: VertexIterator** iterator for the vector of vertices.
- **CGMesh:: FaceIterator** iterator for the vector of faces (or a vector of faces).
- **vcg:: tri:: Allocator <CGMesh>:: AddVertices** (*<mesh>, number of vertices*);
- **vcg:: tri:: Allocator <CGMesh>::AddFaces** (*<mesh>, number of faces*);

3.2.5 Bounding Box

A bounding box is defined in VCG as an object of type Box3 (vcg:: Box3). The inclusion to do in order to use the box is as follows:
# include <vcg/space/box3.h>
Now you can instantiate objects of type Box3 and use their functions. If **B** is an object of type Box3 its main functions are the following:

- **b.min** is the CGPoint the minimum bounding box
- **b.max** is the maximum of CGPoint bounding box
- **b. Center ()** returns a CGPoint of Center of the bounding box.
- **b. Diag ()** returns the length of the diagonal of the bounding box. The type of the return value is dependent on the type of bounding box.
- **b. DimX ()** returns the length of x in bounding box.
- **b. DimY ()** as above, but for y
- **b. DimZ ()** as above, but for the z
- **b. setNull ()** set the box as a null.
- **b. Add (CGPoint)** change the bounding box in agreement with the point passed as a parameter. If the point is within the bounding box, nothing happens.
- **b. Add (Box3)** change the bounding box in agreement with the box passed as a parameter. If the box is within the bounding box, nothing happens.
- **b. IsIn (CGPoint)** returns a Boolean. True if the point is inside the bounding box, False otherwise. The extremes of box are included. **b. IsInEx (CGPoint)** returns a Boolean. True if the point is inside the bounding box, False otherwise.
- **b. Collide (Box3)** returns True if the two boxes intersect.
- **b. P (i)** returns the i-th vertex of the box. The vertices are CGPoint and go 0 to 7.
- **b. Volume ()** returns the volume of the box.

4. The project

4.1 Background

The project is an extension of the thesis of student Daniele Zuddas (Università degli studi di Cagliari). His thesis tried to implement a system to obtain the profile of a vessel in 2 dimensions (see Figure 16) from a real vessel with the help of a laser.

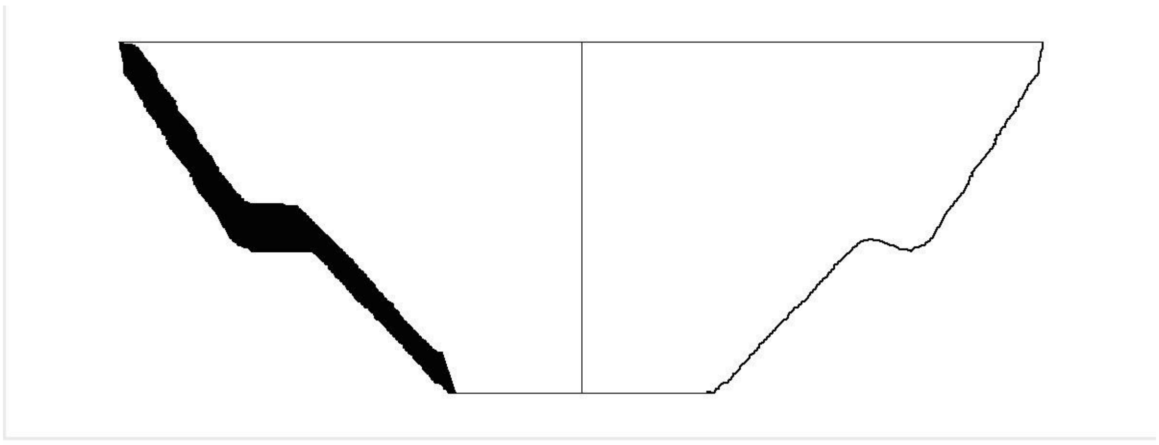

Illustration 15: Vessel's profile.

The project's objective is, from the output of Daniele Zuddas's thesis (vessel's profile in 2D), to do a virtual reconstruction of the original vessel in 3 dimensions using the graphics visualization software CGView.

4.2 Software

The software we are working with is the graphic's viewer CGView. Based on the VCG library for mesh processing and the Qt Toolkit for its interfaceand developed by Computer Graphics Group of the University of Cagliari. The goal of the project consists on making a plugin of this viewer to visualize a vessel in 3D from its profile. This plugin is called "Lathe", (see Section 3, CGView).

4.2.1 Premises

One of the problems of automatic representation of vessels is the infinite variety of vessel's types that exist and the particularity that each one of them can show. We must not forget that these pieces are made by hand, so they may have unique characteristics, and that they are really complex for the archeologist to represent them.

For this reason, this version of the representation's automatic system has been made by having in mind these premises:

- The vessel is completely circular.
- Particular irregularities are not represented.
- The decoration is ignored.

These premises decrease a lot the complexity of the project, and its typology can be visualized more clearly.

4.2.2 Functions implemented

To modify our Mesh or input, they must be created new vertices and create connections between them (faces). To achieve this, some functions have been implemented:

- *InitialRadix:* Function to determinate the vessel's radius. It will normally be half the height and half profile.

- *DrawBase:* Function to draw vessel's base, creating a number of faces at the base so it stays closed.

- *AddVertices:* Function to create the new vertices of our Mesh. Add 360 vertices for each one which was in the beginning, to make them rotate 360 degrees to form a circle entire.

- *AddFaces*: Function to create the different faces of our new Mesh. A face is formed by the union of 3 vertices.

- *UpdateBoundingBox:* we need to add with CGView's function "bbox.Add() all vertices that were created for our mesh".

- *AdequateLight:* To have a good visualization of our Mesh, we will disable the light with "disable (GL_LIGHT)" function and enable BLEND to have a good view of the vessel.

- *CaracteristicPoints:* Function to calculate the diferents points where the vessel's curvature changes, (see 4.2.4 Characteristics points).

4.2.3 Reading input

The reading of the input is made from a ".ply" file which represents a number of points in space that draw a vessel's profile, representing the output of Daniele Zuddas's thesis. Knowing that this profile is in 2D we will work on the plane (X, y), leaving the Z coordinate to 0.

.Ply format data are formed in the following way:

```
ply
format ascii 1.0
comment created by MATLAB
ply_write element vertex 200
property float x
property float
and property
float z
end_header
-4.460273 -26.451251 0
-4.467952 -26.267871 0
-4.475163 -26.085980 0
-4.477052 -25.919093 0
-4.479254 -25.751337 0
...
```

When we open the .ply file with CGView it's showed the initial vessel's profile, (see Illustration 17).

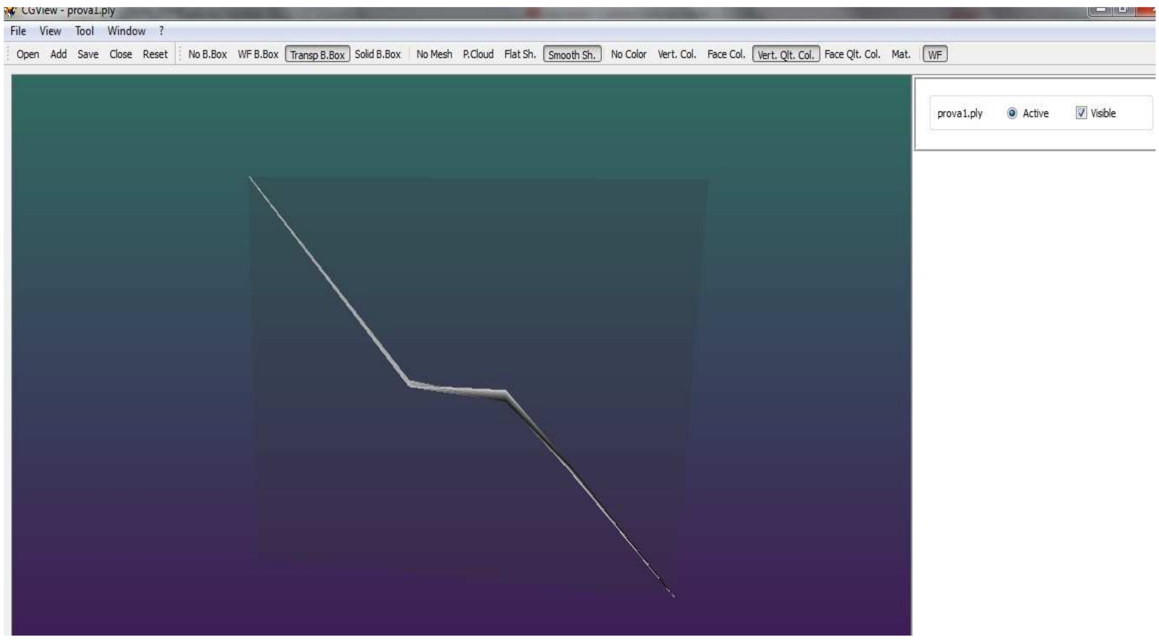
CGView - prova1.ply
File View Tool Window ?
Open Add Save Close Reset
No B.Box WF B.Box Transp B.Box Solid B.Box No Mesh P.Cloud Flat Sh. Smooth Sh. No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat. WF
prova1.ply Active Visible

Illustration 16: Reading input vessel's profile with GCView.

4.2.4 Characteristic points

The calculation of profile's characteristic points problem is a trivial problem. We can't calculate the discontinuity points in a mathematic sense, since there is no way to calculate exactly when the curve makes an interesting change.

Another factor making harder this complex calculation is that vessels are made by hand, so they don't follow any mathematical rule and can be highly irregular. The input is a vector of points and not a mathematic function so it's more difficult to make the analysis. The characteristic point concept is questionable and relative to archeologist's experience who follows the design or the context where the vessel has been found.

For these reasons it has been decided to make the difference between the characteristic points on an empirical way. This methodology has the advantage of being simple and having the safety of working pretty well for certain types of interest points. But it has the disadvantage that since it's not a strong mathematical theory, it might fail for some specific profiles.

Some typical characteristics in a vase are the following ones, (see Illustration 18):

- *EP*: Vessel's limits: high and low points of the object. These are the first vertices, the last ones and the midpoint on the Y coordinate.
- *VT:* Vertical tangents: points of maximum or minimum curvature.
- *IP*: Indexing points: points where the curvature's meaning changes.
- *CP*: Corner: exchange points of pronounced curvature.

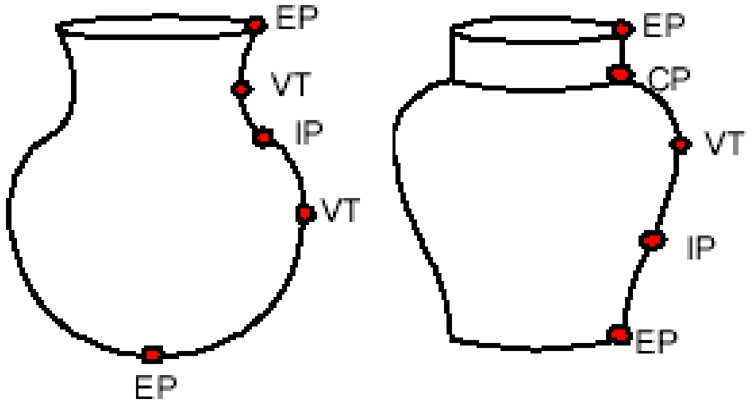

Illustration 17: Characteristic points of vessel

4.2.5 Plotting

To see our vessel in 3D, we must revolutionize each one of the profile’s vertices (input) regard an axis of rotation and a certain radius. To assign this rotation center, we must do a translation on the x-axis of the vessel’s base. The value of this translation will be the vessel’s radius (since we have no details about the original radius and there are many different types of vessels, we have provided the initial radius as half of its height). Once determined the radius, it must be done a revolution of all profiles vertices respect to the axis of rotation this way:

For each profile vertices create a new vertex while keeping the coordinate Y but modifying the X and Z, (see Appendix B).

The faces creation is done by putting together 3 vertices (triangular faces).Each face must have a color with the function vcg:Color4b(), this function must have 4 parameters Float type in RGBA format. A variant of the 'brown' color has been selected for the faces with the values in RGBA = (215,180,50,0). If the vertex is a characteristic point, we proceed to draw the face with black color RGBA = (0,0,0,255) to differentiate it from the other points.

Once created all vertices and their matching faces we proceed to draw the vessel’s base creating different faces with the lower object’s vertices.

When the development of the object it’s done, the BoundingBox gets updated, adding all vertices created with the function ' bbox.Add()'.

Finally, the light and camera parameters will be changed to see more clearly the resulting object with the GL_BLEND() and GL_LIGHT() functions, (see Appendix C).

4.2.6 Results

The plugin ‘Lathe’ generates a 3D image where you can view the reconstructed vessel with its corresponding characteristic points. CGView allows viewing only the vertices (see illustration 19 and 20) or with all their different faces drawn (see illustrations 21 and 22) with their corresponding color.

To see the plugin’s operation, we must run the CGView software and open the '. ply' file with the vessel’s profile to be treated. Once opened, select the tab Tool - > Lathe to run our plugin.

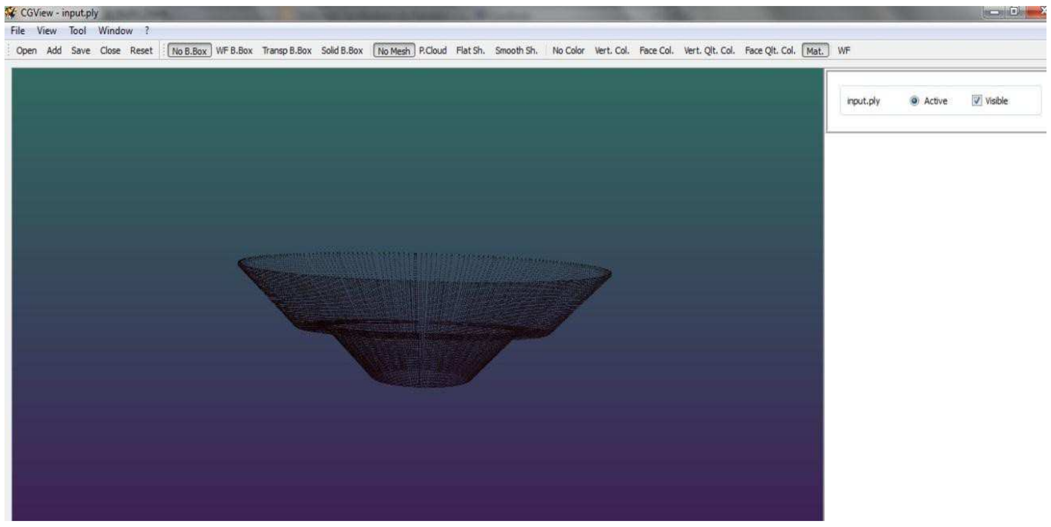
CGView - input.ply
File View Tool Window ?
Open Add Save Close Reset No B.Box WF B.Box Transp B.Box Solid B.Box No Mesh P.Cloud Flat Sh. Smooth Sh. No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat. WF
input.ply Active Visible

Illustration 18: Phiale with their vertices

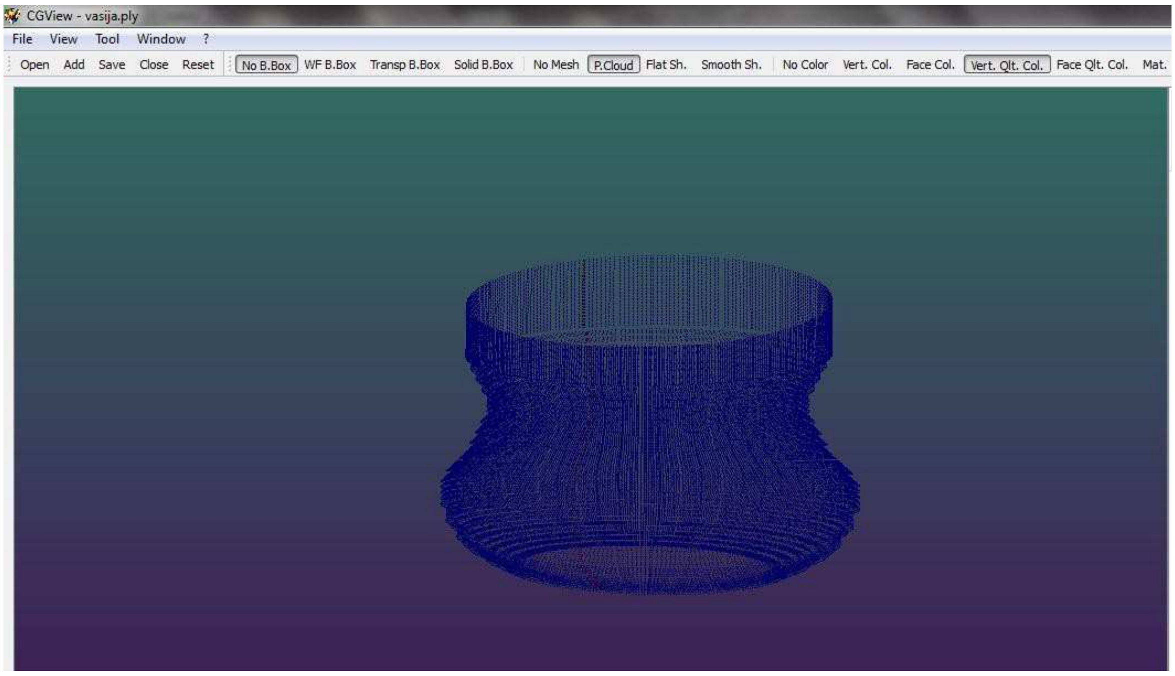
CGView - vasija.ply
File View Tool Window ?
Open Add Save Close Reset No B.Box WF B.Box Transp B.Box Solid B.Box No Mesh P.Cloud Flat Sh. Smooth Sh. No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat.

Illustration 19: Amphora with their vertices

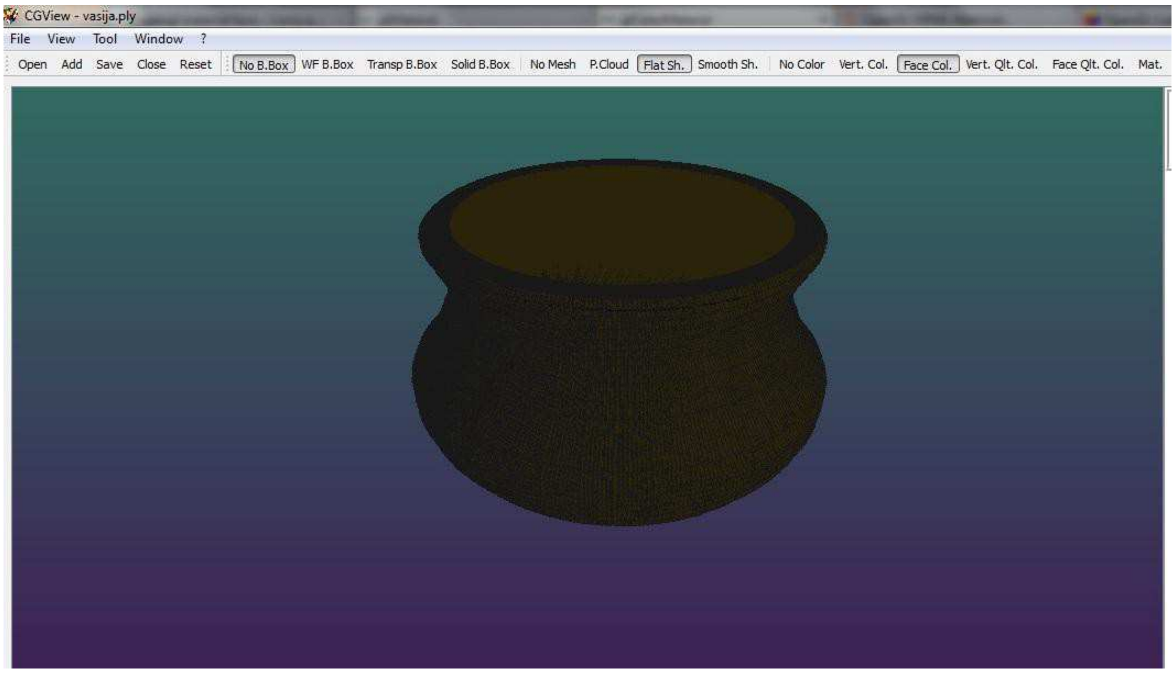
CGView - vasija.ply
File View Tool Window ?
Open Add Save Close Reset No B.Box WF B.Box Transp B.Box Solid B.Box No Mesh P.Cloud Flat Sh. Smooth Sh. No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat.

Illustration 20: Amphora's reconstruction

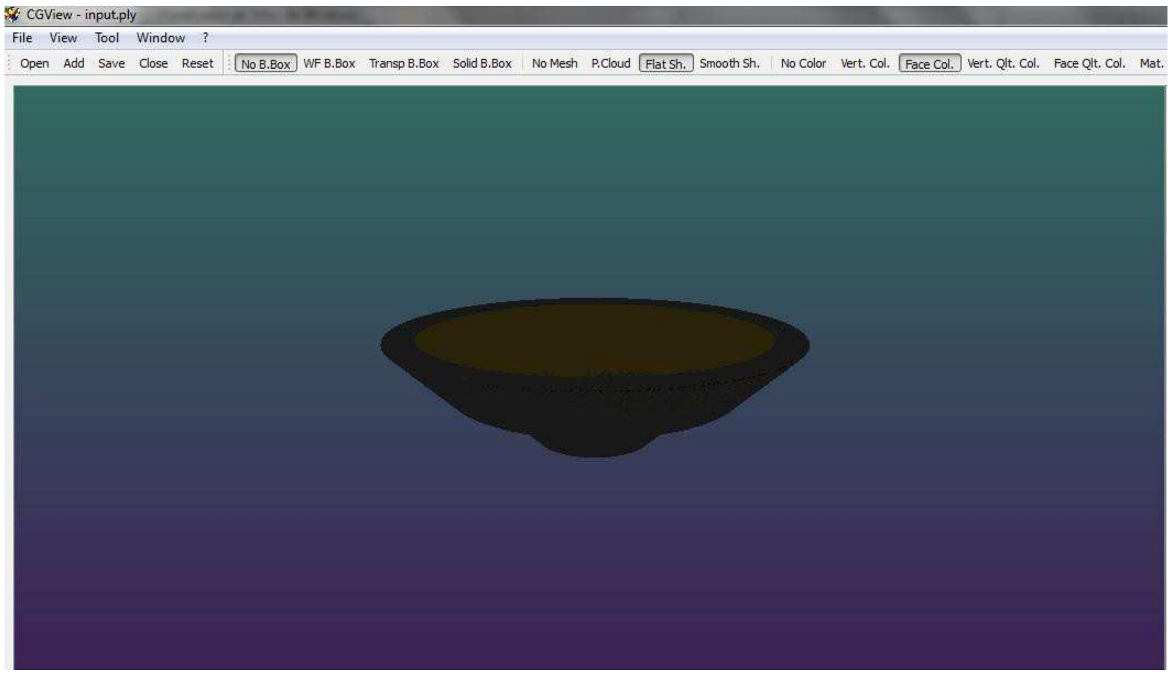
CGView - input.ply
File View Tool Window ?
Open Add Save Close Reset No B.Box WF B.Box Transp B.Box Solid B.Box No Mesh P.Cloud Flat Sh. Smooth Sh. No Color Vert. Col. Face Col. Vert. Qlt. Col. Face Qlt. Col. Mat.

Illustration 21: Phiale’s reconstruction

.

5. Summary and future developments

5.1. Final assessment

This first software's version solves successfully the initial problem proposed, and his design is in good quality to visualize correctly the basic structure of how the original vessel would be given his profile. The function of the characteristic points calculation isn't fully developed and may give some errors to detect points in some special vessel.

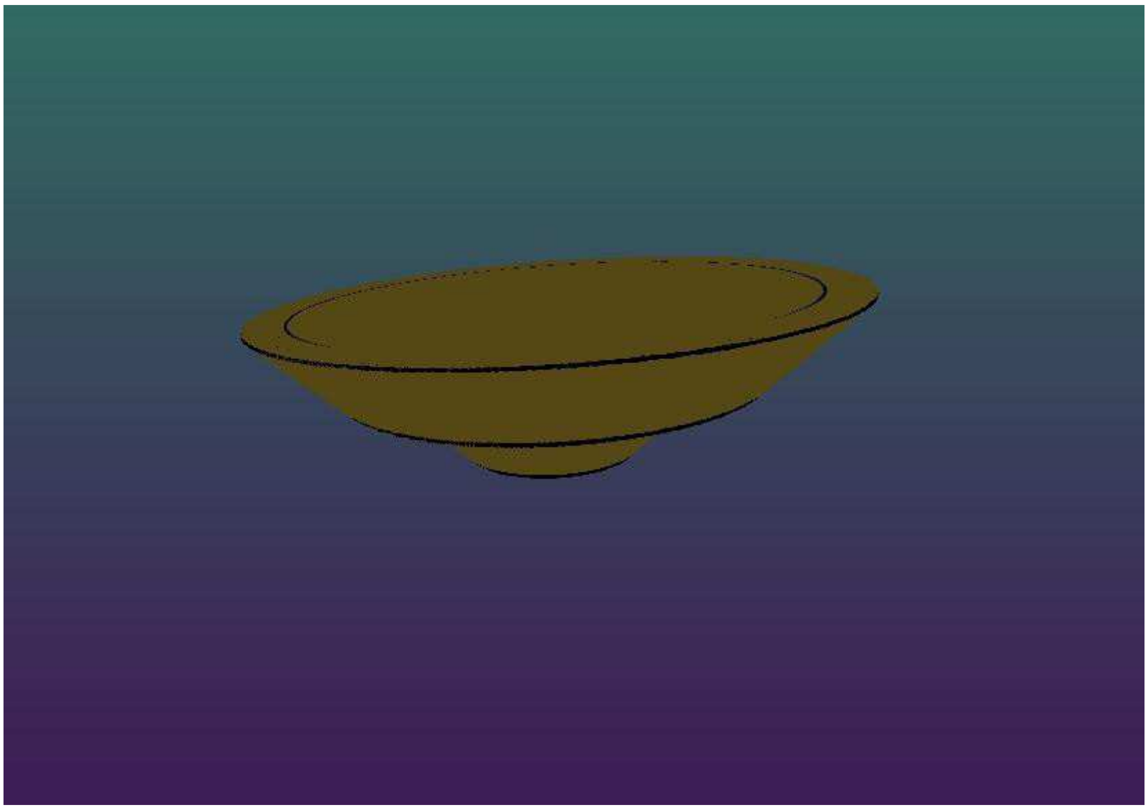

Illustration 22: Vessel's reconstruction with Characteristic Points

5.2. Potential improvements or developments

A possible upgrade will be modifying the characteristic points function to give a correct result for all types of vessels and all the characteristics points. The calculation of vessel's base, which we have specified that are a perfect circle, can also be improved because there're vessels whose bases are elliptical.

It should be implemented a system which would make a circular or elliptic representation of the vessel depending on the profile. The vessel's radius revolution should be stipulated as well since with only the profile's parameters it's impossible to determinate which one is the original vessel's radius.

Another possible software development is creating a database to register different types or families profiles, so we can obtain more details about the shape, dimensions, decoration or material from a profile, and obtain more entry data to make a more realistic 3D representation. With this database we could do statistical studies of different vessel's types that exist and recognize their specific properties.

An plugin improvement it's to do the vessel's reconstruction from a vessel's piece in 3D (sherd), where the input should be a ".ply" format file but with real data from the 3 coordinates X, Y and Z. If we have data of the vessel's curvature, it's possible to make a real estimation of the rotation axis and his radius (see Appendix A).

There are many evolutions and different changes that can be done to the plugin and the project, but for now is expected to create a second version that deals with the classification problem of archaeological ceramics through others software development faster and accurate than this version to provide an useful tool for those that are working in the study of the history.

Appendix A

Estimation of the axis of rotation of sherd pottery: a multistep model based approach

A.1 A model of a pottery fragment

During the classification of a pottery fragments, archaeologists assume that original pots were made on a potter's wheel. Regarding this constraint, ancient pots can be modeled as *radially symmetric objects*. Such objects and its fragments have two important geometrical properties demonstrated in Illustration 26:

a) Normals of the object/fragment surface go through the axis of rotation

b) A plane perpendicular to the rotation axis intersects the object in a circle and the fragment in a circular arc with the same center lying on the axis

Under the assumption that the original pots were radially symmetric, fragments of archaeological pottery can be described by the same model. The geometrical properties of a fragment do not depend on its position, thus they can be used for an estimation of an appropriate axis of rotation. And this is the main idea of the estimation process proposed in the following sections.

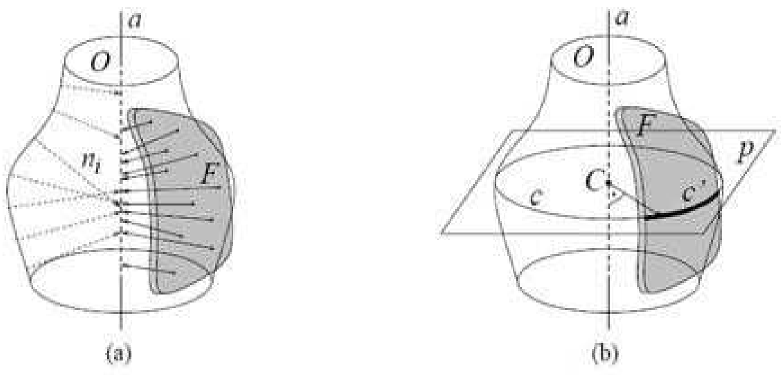

Illustration 23: A model of a radially symmetric object *O* with the axis of rotation *a* and its fragment *F*: a) surface normals *ni*, go through the axis *a*. b) a plane *p* perpendicular to the axis *a* intersects the object *O* in a circle *e* and the fragment *F* in a circular arc *e'* with the same center *C* lying on the axis.

A.2 An estimation of an initial position of the axis of rotation

In the beginning of the estimation process, an initial position of the axis of rotation is determined. The estimation assumes radial symmetry of ancients pots and exploits the first property of radially symmetric objects: *surface normals go through the axis of rotation of the object*. There exist two different approaches based on this fact: Hough transform and numerical optimization. The Hough transform inspired method is robust against outliers, but it lacks in speed and accuracy. Regarding that, an *optimization approach* is used for the initial estimation. The initial position of the axis is obtained using a standard least squares approach as the line which minimizes the following objective function:

(1)
$$\min_{a} \sum_{i=1}^{M} d^2(n_i, a) \quad ,$$

Where M is the number of surface normals and *d(ni,a)* is the Euclidean distance between the normal *ni* and the axis *a*. The distance can be expressed as the length of the transversal line between the particular normal *ni = Xi + ti · Ni* given the point *Xi* and the normal direction *Ni*, and the rotation axis *a = Xo + to · No* given by the point *Xo* and the normal direction *No:*

(2)
$$d(n_i, a) = \frac{(X_i - X_0) \odot (N_i \times N_0)}{||N_i \times N_0||} \quad ,$$

Where the operators · and *X* denote dot and vector product of two vectors, respectively, and ||.|| is the length of a vector. If the normals *Ni* and *No* are parallel, the equation is reduced to:

(3)
$$d(n_i, a) = ||(X_i - X_0) \times N_0|| \quad .$$

The optimization problem is solved for the unknown rotation axis *a* given by the point *Xo* and the normal direction *No*. It can be shown that the optimal position of the point *Xo* depends on the direction *No*. Regarding that, an initial position of the axis of rotation can be determined by an optimization process which runs over only two dimensional searching space. Such estimation is effective, fast and reliable.

A.3 A robust estimation of the axis of rotation

In the initial estimation step, a position of the axis of rotation is estimated by a direct least squares minimization. This type of optimization approaches is known to be sensitive to outliers. Statisticians have developed various sorts of robust statistical estimators. The most relevant class for an estimation of parameters is so called *M-estimators*. Mathematical theory of *M-estimators* is available in many books. The basic idea of robust estimation is to reduce the influence of outliers by replacing the squared residuals in the standard least squares minimization

(4) $$\min \sum_i r_i^2$$

by another less increasing function *p* of the residuals, yielding

(5) $$\min \sum_i \rho(r_i) \ .$$

Instead of solving directly, the minimization can be implemented as an iterative re-weighted process

(6) $$\min \sum_i w(r_i^{(k-1)})\, r_i^2 \ ,$$

Where *w* is so called *weight function* and the superscript *(k)* indicates the iteration number. There exist a wide range of weight functions with different properties. A very popular among statisticians is *Huber's function:*

(7) $$w(r_i) = \begin{cases} 1 & \text{if } |r_i| \le c\hat{\sigma} \\ \dfrac{c\hat{\sigma}}{|r_i|} & \text{otherwise} \end{cases} ,$$

Where is a robust standard deviation of residual errors and *c* is a tuning constant with value *c = 1.345*. Another widely used weight function is *Turkey's biweight*

(8) $$w(r_i) = \begin{cases} \left[1 - \left(\dfrac{|r_i|}{c\hat{\sigma}}\right)^2\right]^2 & \text{if } |r_i| \le c\hat{\sigma} \\ 0 & \text{otherwise} \end{cases}$$

with the tuning constant *c = 4.6851*. The robust standard deviation of residual errors can be estimated as

(9) $$\hat{\sigma} = 1.4826 \operatorname*{median}_{i} |r_i|$$

An application of the M-estimators for a robust determination of the axis of rotation is straightforward and it leads to the following algorithm:

1. Get an initial position of the axis *a = Xo + to · No* by the direct least squares minimization of Eq. 1 with distances given by Eq. 2 and Eq. 3

2. Use distances from the currently estimated axis a as residual = *d(ni, a)*

3. Estimate the robust standard deviation of the residual (Eq. 9)

4. Compute weights from residual by the chosen weight function *w* (for example Eq.7 or Eq. 8)

5. Estimate a new position of the axis *ã* which minimizes the weighted least squares problem Eq. 6

6. Convergence check: if, set *a = ã* and go back to the step 2

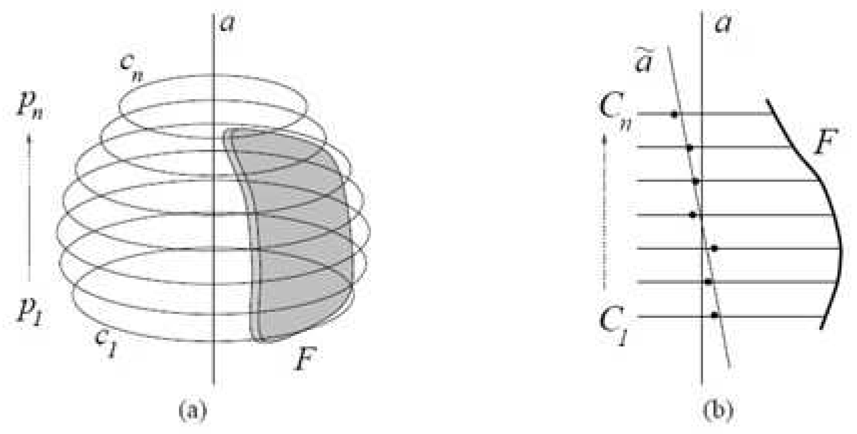

Illustration 24: An estimation of the axis of rotation by a circle/line fitting: a) Circle fitting. b) Line fitting

As can be seen, the algorithms an iterative reweighted modification of the initial direct least squares minimization. Regarding that, it can be implemented very easily on top of the original estimation. The application of M-estimators preserves efficiency and reliability of the computation and adds robustness against outliers in the data and stability to systematic errors during processing of fragments.

A.4 An iterative refinement of the estimated rotation axis

The previous estimation steps are based on property of surface normals. The problem is that the normals are typically not known in advance and they have to be determined first. During the determination, various factors (such as noise, outliers and systematic errors) degrade estimates of the normals and, consequently, the whole estimation of the axis of rotation. Regarding that, the position of the axis can be further improved by another, nonnormal based, estimation method.

To estimate the axis of rotation of radially symmetric objects without knowledge on their surface normals, the second property of such objects can be exploited: *a plane perpendicular to the rotation axis intersects the object in a circle and its fragment in a circular arc with the same center lying on the axis*. In our case only an approximation of the axis is known from the previous estimation steps, the original method cannot be used directly. Instead, the axis of rotation is improved by the following iterative refinement process:

1. Get an initial position of the axis of rotation *a* by the optimization approach described in the previous sections

2. Generate a set planes *pi* perpendicular to the axis *a*.

3. For every plane *pi*:
    a) Detect an intersection between the plane *pi* and the fragment *F*
    b) Fit the detected intersections by a circle *ci*
    c) Estimate a center *Ci* of the fitted circle *ci*

4. Estimate a new position of the axis ã as the line which approximates the centers *ci.*

5. Convergence check: if, $||\tilde{a} - a|| > \varepsilon$ set *a* = *ã* and go back to the step 2.

The algorithm assumes that the detected intersections are circular arcs. With respect to the fact that the position of the axis of rotation can be inaccurate in the beginning, such assumption can be wrong. Thus the algorithm has to cope with distorted intersections and their influence on the circle and line fitting phases (steps 3b and 4).

Robustness of these fittings is achieved by an application of M-estimators in the same straightforward way as was used for the robust estimation of the axis of rotation described in the previous section.

Appendix B

Circle drawing

A **circle** is a simple shape of Euclidean geometry consisting of those points in a plane that are a given distance from a given point, the center. The distance between any of the points and the center is called the radius.

Circles are simple closed curves which divide the plane into two regions: an interior and an exterior. In everyday use, the term "circle" may be used interchangeably to refer to either the boundary of the figure, or to the whole figure including its interior; in strict technical usage, the circle is the former and the latter is called a disk.

A circle can be defined as the curve traced out by a point that moves so that its distance from a given point is constant.

To create a circle from a vertex with our software first of all I am going to create a typedef to hold our circle properties as we call them.

```
typedef struct
{
```
This will hold the x values as we need them
```
float x;
```
This will hold the y values as we need them
```
float y;
```
Call it CIRCLE
```
}CIRCLE;
```

Then I am setting the property circle to act like the typedef CIRCLE
```
CIRCLE circle;
```

Now for the meat of this tutorial, the circle creation function
The properties this takes are:
k – the translation on the y axis
r – the radius of the circle
h – the translation on the x axis

```
void createcircle (int k, int r, int h) {
```
Now we begin drawing our lines
```
glBegin(GL_LINES);
```
Here we are setting up the vertices all in one line function; this sets every 2 vertices to a line.

First we begin our loop, this loop will cycle through, constantly changing our X and Z values for our lines. It cycles through until it reaches 360, increasing in intervals of 1
```
for (int i = 0;i < 360;i++)
{
```

Now to set up the current x value that we need for our vertex I am setting it to the radius of the circle, times by the cosine of the current value of i, then I amtaking h to translate it:
circle.x = r * cos(i) – h;

Then to set up the current *y* value that we need for our vertex, I am setting it to the radius of the circle, times by the sine of the current value of *i*, then I am adding k to translate it:
circle.z = r * sin(i) + k;

Then I am drawing the vertex:
glVertex3f(circle.x + k, circle.z – h,0);

Now for the second part of the line, I am doing the same as above, only I am moving it 0.1 units so that I am not down to points. This also Lowers the chance of any holes occurring in the circle:
circle.x = r * cos(i + 0.1) – h;
circle.z = r * sin(i + 0.1) + k;

Then I am drawing the vertex:
glVertex3f(circle.x + k,circle.z – h,0);
}
glEnd();
}

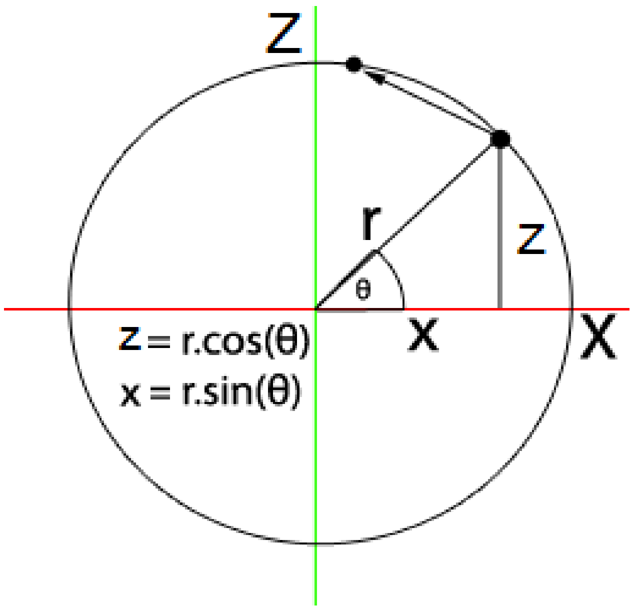

Illustration 25: Variation of X and Z coordinates.

Appendix C

OpenGL functions

C.1 glLight

glLight sets the values of individual light source parameters. *Light* names are a symbolic name of the form GL_LIGHT *i*, where *i* ranges from 0 to the value of GL_MAX_LIGHTS - 1. *pname* specifies one of ten light source parameters, again by symbolic name. *params* is either a single value or a pointer to an array that contains the new values.

To enable and disable lighting calculation, call to *glEnable* and *glDisable* with argument GL_LIGHTING. Lighting is initially disabled. When it is enabled, *light* sources that are enabled contribute to the lighting calculation. Light source i is enabled and disabled using *glEnable* and *glDisable* with argument GL_LIGHT *i*.

The seven light parameters are as follows:

GL_AMBIENT

params contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive performable value maps to 1.0, and the most negative performable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0, 0, 0, 1).

GL_DIFFUSE

params contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive performable value maps to 1.0, and the most negative performable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for GL_LIGHT0 is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 1).

GL_SPECULAR

params contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive performable value maps to 1.0, and the most negative performable value maps to -1.0.Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for *GL_LIGHT0* is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 1).

GL_POSITION

params contains four integer or floating-point values that specify the position of the light in homogeneous object coordinates. Both integer and floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The position is transformed by the *modelview* matrix when *glLight* is called (just as if it were a point), and it is stored in eye coordinates. If the w component of the position is 0, the light is treated as a directional source. Diffuse and specular lighting calculations take the light's direction, but not its actual position, into account, and attenuation is disabled. Otherwise, diffuse and specular lighting calculations are based on the actual location of the light in eye coordinates, and attenuation is enabled. The initial position is (0, 0, 1, 0); thus, the initial light source is directional, parallel to, and in the direction of the -z axis.

GL_SPOT_DIRECTION

params contains three integer or floating-point values that specify the direction of the light in homogeneous object coordinates. Both integer and floating-point values are mapped directly. Neither integer *nor floating-point* values are clamped. The spot direction is transformed by the upper 3x3 of the *modelview* matrix when *glLight* is called, and it is stored in eye coordinates. It is significant only when GL_SPOT_CUTOFF is not 180, which it is initially. The initial direction is (0,0,-1).

GL_SPOT_EXPONENT

params is a single integer or floating-point value that specifies the intensity distribution of the light. Integer and floating-point values are mapped directly. Only values in the range [0,128] are accepted.

Effective light intensity is attenuated by the cosine of the angle between the direction of the light and the direction from the light to the vertex being lighted, raised to the power of the spot exponent. Thus, higher spot exponents result in a more focused light source, regardless of the spot cutoff angle (see GL_SPOT_CUTOFF, next paragraph). The initial spot exponent is 0, resulting in uniform light distribution.

GL_SPOT_CUTOFF

params is a single integer or floating-point value that specifies the maximum spread angle of a light source. Integer and floating-point values are mapped directly. Only values in the range [0,90] and the special value 180 are accepted. If the angle between the direction of the light and the direction from the light to the vertex being lighted is greater than the spot cutoff angle, the light is completely masked. Otherwise, its intensity is controlled by the spot exponent and the attenuation factors. The initial spot cutoff is 180, resulting in uniform light distribution.

C.2 glBlend

In RGBA mode, pixels can be drawn using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the frame buffer (the destination values). Blending is initially disabled. Use glEnable and glDisable with argument GL_BLEND to enable and disable blending.

glBlendFunc defines the operation of blending when it is enabled. sfactor specifies which method is used to scale the source color components. dfactor specifies which method is used to scale the destination color components. The possible methods are described in the following table. Each method defines four scale factors, one each for red, green, blue, and alpha. In the table and in subsequent equations, source and destination color components are referred to as (R_s,G_s,B_s,A_s) and (R_d,G_d,B_d,A_d). The color specified by glBlendColor is referred to as (R_c,G_c,B_c,A_c). They are understood to have integer values between 0 and (k_R,k_G,k_B,k_A), where and (m_R,m_G,m_B,m_A) is the number of red, green, blue and alpha bitplanes.

Source and destination scale are referred to as (s_R,s_G,s_B,s_A) and (d_R,d_G,d_B,d_A). All scale factors have range [0,1].

Bibliography

[1] T.Davis, J.Neider, and M.Woo, *OpenGL Programming Guide.* Addison-Wesley, 1993.

[2] D.Zuddas, *“Clasificazione automatica di cocci di vasi antichi”.* Cagliari, 2009

[3] M.Kampel and R.Sablatnig, “*Computer aided classification of ceramics”* in VAST’00, 200.

[4] F.Melero, A.León, F.Contreras, and J.Torres, “*A new system for interactive vessel reconstruction and drawing”* April 2003

[5] C.Maiza, “*Classification d’objets de révolution: application aux poteries sigillées”* . Toulouse, December 2008

[6] Mara & Sablating. “*Determination of ancient manufacturing techniques of ceramics by 3D shape estimation*”. 2006

.