Universidad de Granada

Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos

Alhambra: a model for producing 2D animation

Thesis

Domingo Martín Perandrés

Granada, 1999

Acknowledgements

I want to express my acknowledgement to all people who have hold up, suffered, supported, and back me during these years, some of them specially hard. Thanks specially to Maribel for ALL. Thanks to Juan Carlos for his trust and help. Thanks to Papá, Mamá, Rosi, Maria, Ernesto, Abuelas, Carlos, Manolo. Thanks to Rafa. Thanks to Lola. Thanks to Daniela Tost and Pere Brunet for their support. Thanks to all my friends and workmates.

Thanks to the tribunal members and all those who can finish of reading the memory. I hope it can be used for something useful.

Thanks to everybody for their ideas and corrections.

Thanks to people that has developed software (Linux, TEX, LATEX, and so on) that has been done free for being used for everybody, as myself.

to Maribel and my parents

Contents

List of Figures

Prologue

From the beginning of computer graphics there has been a great interest in producing very realistic images. The main goal has been to reproduce with high fidelity, from a mathematic and physics point of view, the behavior of related elements with the production of an image. The study of the physics of the interaction between the light and objects has carried out to ray tracing and radiosity methods. But there is also a great interest in obtaining a real appearance in movement, studying the application of dynamic and cinematic. Currently there is research work for obtaining virtual beings that can sense their world and react as result of what happens in them. This investigation line can be called realistic visualization.

The other line, which can be called non realistic or expressive visualization, has been developed in the last few years. The objectives are different: to produce images that transmit qualitative visual information instead of quantitative; expressing, not representing; showing what it is, not how it is.

The classical animation films can be included in this last category. Many people considers that it is an art, and it needs very big teams of persons and material for producing feature films. However, it is a powerful industry with great capacity for selling its products.

From its apparition, the computer has been integrated as a tool for many processes, which has improved production. This does not totally happen in classic animation.

Given the basic elements, we exposed the final goals of this memory: to obtain a system that allows to produce animation that looks like classical animation, in all aspects (visual, expressive, etc.). The solution that is presented is the *Alhambra* model.

This memory is a resume of the full version in spanish. Howewer, the most reduce part is the introduction chapter, specially the history and the phases for producing classical animation, as well as the classic techniques. The classification section, our objectives and the rest of sections are almost complete, eliminating only some minor details. The structure of the document is the same as the original, including chapters, sections, and so on, and only one figure has been eliminated.

The history of animation is presented in first chapter. Then, there is a classification, and the capabilities of an ideal system are presented. Finally, the previous works are commented.

In second chapter, there is an exposition of the elements of the proposed model,

in relation with the visualization process and the animation process.

Chapter three is dedicated to present the implementation of the *Alhambra* model when is applied to a 2-manifold triangle based model. The animation is implemented using functions.

In chapter four, conclusions and future works are presented.

The data formats are in Appendix A.

Chapter 1

Introduction

Animation means to "give life to something without it"[1]. In the context of graphics, it can be understood as the creation of the illusion that things are alive, that they change. This definition not only includes which is normally understood as animation, that is, the illusion of movement when images are projected at some speed, but also includes other possibilities of change, for example, morphing [64], form changes [4] (Figures 1.1, 1.2), lighting changes (Figure 1.3), and so on.

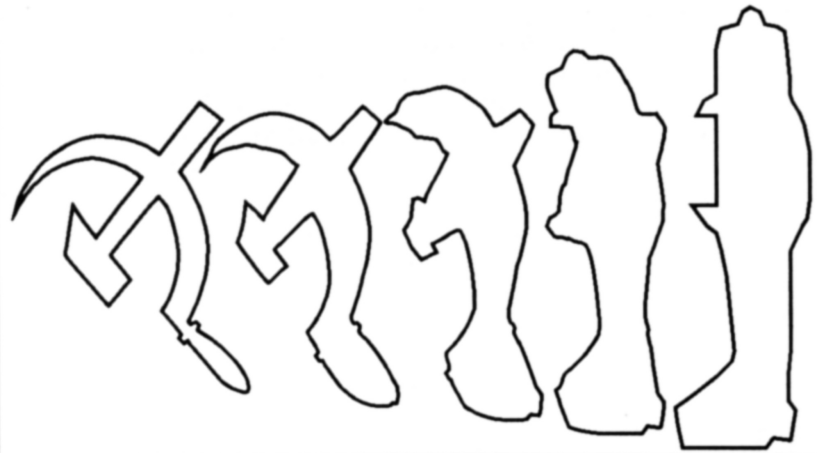

Figure 1.1: Metamorphosis example [64].

1.1 History

A revision of computer generated animation films from 1961 to 1989 is found in [44].

[1] Animation come from Latin word *animus*, that means to give life to something, at the same time, it is related with the Greek word *anemus*, that means breath [36].

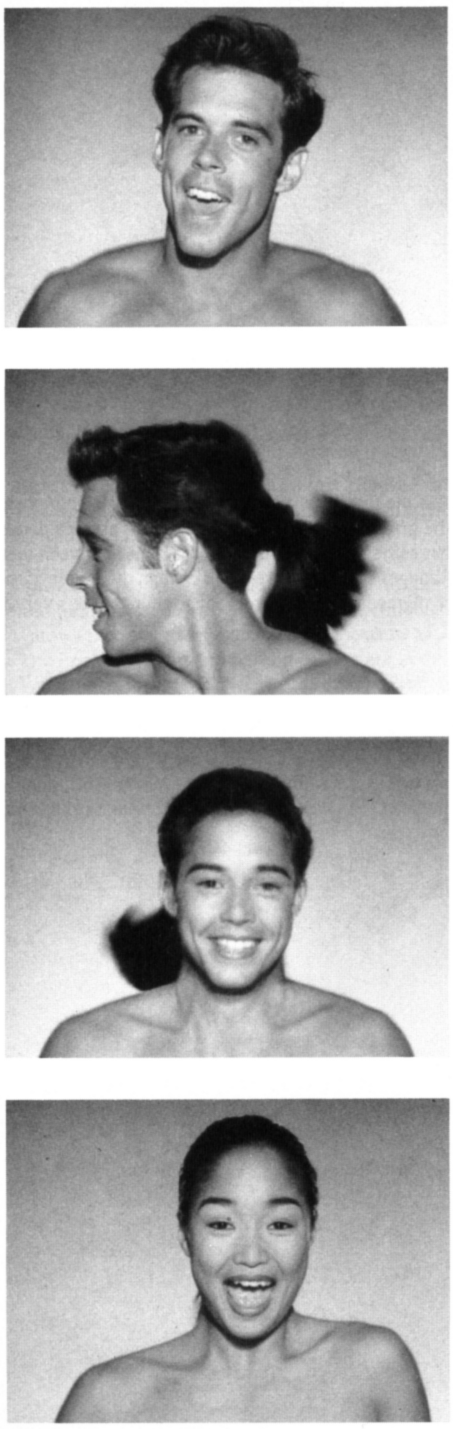

Figure 1.2: Morphing example [4].

1.2 Classification of animation

Several schemes of classification are presented. This is necessary to understand which are the objectives of the thesis. There are several elements that serve for classifying: the agent, person or machine, that produces the animation, the tool used, by hand

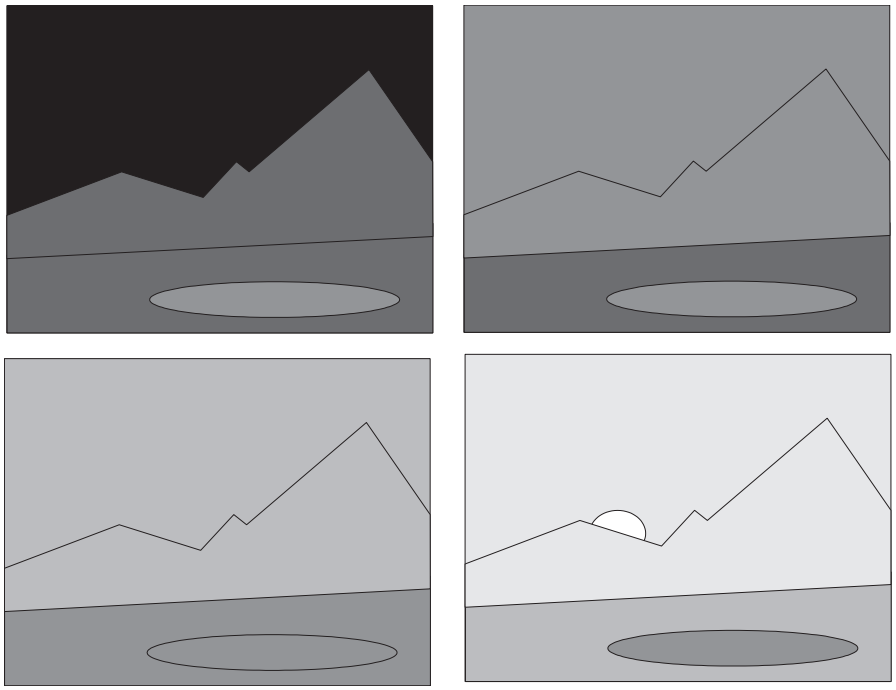

Figure 1.3: Example of a light intensity change.

or automatically, and the visual aspect, amongst others.

The producer can be a person or a machine. if the producer is a person, he or she can or can not use the computer. If the producer is a person and uses the hand we talk about conventional or classical animation. If the person uses a computer, we say that it is computer assisted animation. If the producer is a machine, we say tha is computer or modelled animation Figure 1.4.

Another possibility of classification is the manner the animation is produced: if it is based in drawings, we say that it is key frame animation; if parameters are used, we say that it is parametric or procedural animation. Both concept are explained in page 10.

The visual appearance is other concept for doing the classification (Figure 1.5). We have 2D and 3D appearance animation. In general, classical animation produces 2D appearance animation, while computer animation produces 3D appearance animation.

1.3 Classical animation

In classical animation, the drawer has a mental model of the characters he or she want to represent with draws.

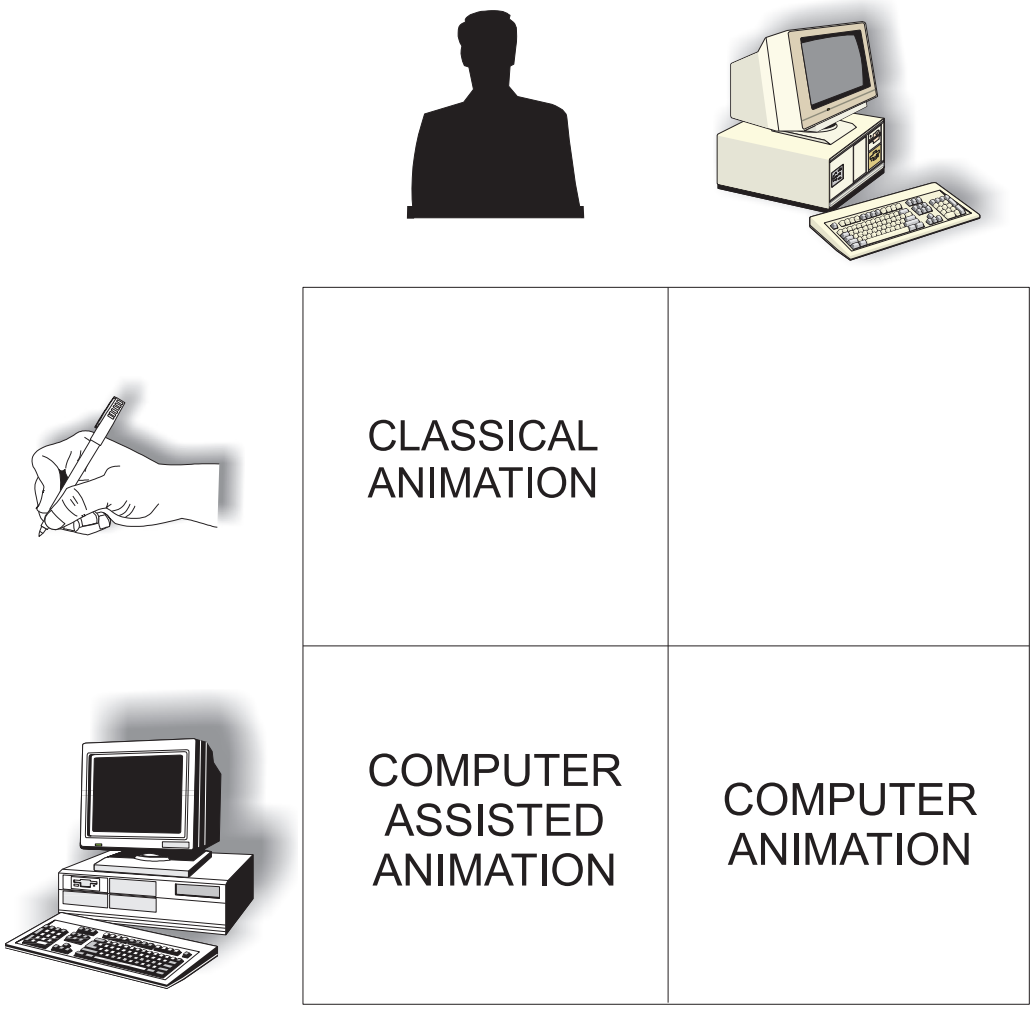

Figure 1.4: Classification of animation based in the producer and the tool.

1.3.1 Production process of classical animation

The production process of an animation film is long and costly. It needs big teams The production has several phases [74].

- Script
- Storyboard
- Sound Track
- Track Breakdown
- Design

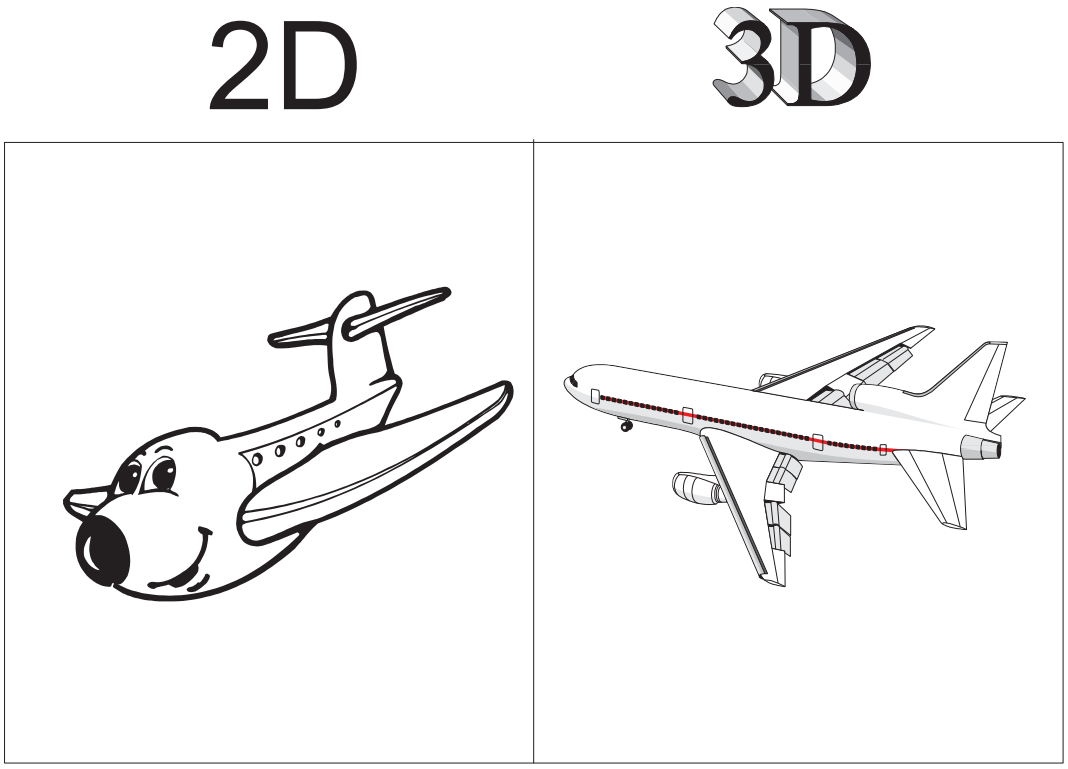

Figure 1.5: Classification of animation based on its visual appearance.

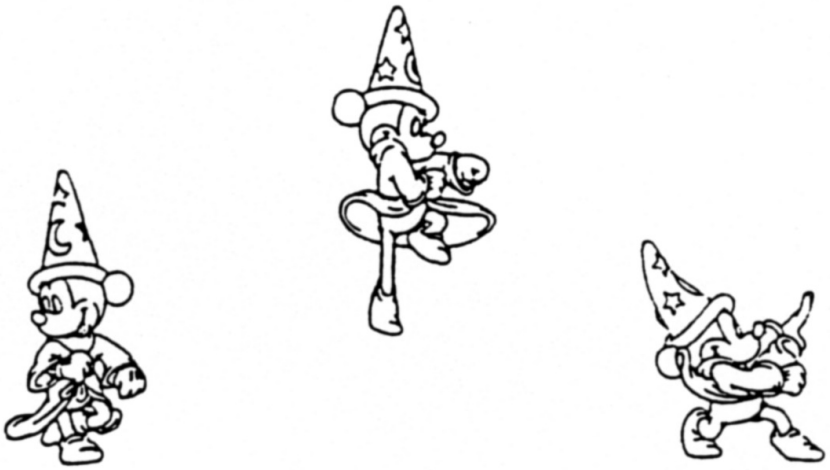

Figure 1.6: Key frames.

- Leica Reel
- Line Tests
- Clean Up
- Trace and Paint

- Backgrounds
- Checking
- Final Shoot
- Rushes
- Dubbing
- Answer Print

1.3.2 Principles of classical animation

From the 20th to the 30th, the animation passed from being a novelty to become an art in the Walt Disney studies. Several techniques appeared that produced a more expressive animation. These techniques were improved and became fundamental rules of the animation. These rules are:

1. Squash and Stretch (Figure 1.8).
2. Timing
3. Anticipation (Figure 1.9).
4. Staging
5. Follow Through and Overlapping Action
6. Slow In Slow Out
7. Arcs
8. Exaggeration

Figure 1.7: Key frame inbetweening.

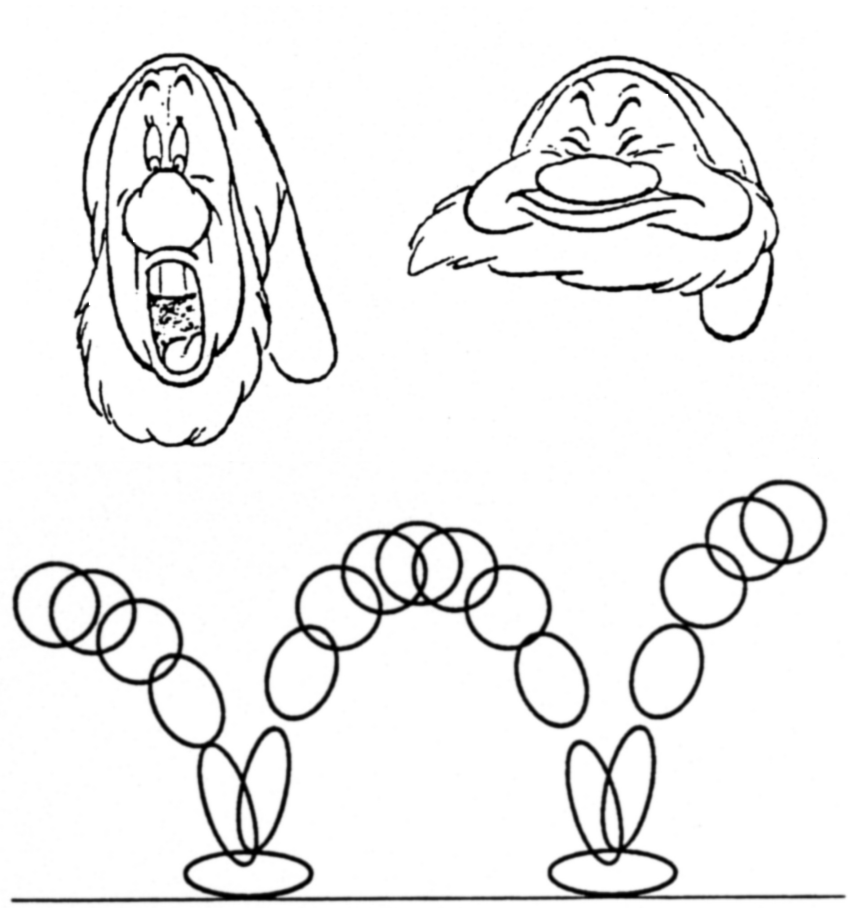

Figure 1.8: Squashing and stretching technique [55].

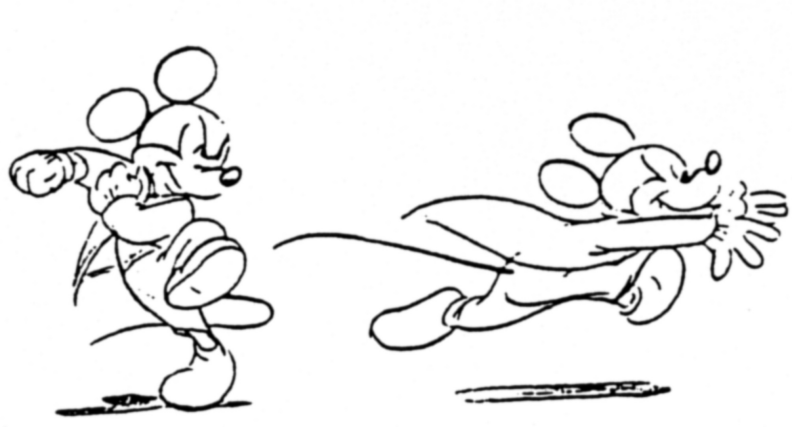

Figure 1.9: Anticipation technique.

9. Secondary action

10. Appeal

These techniques, developed for classical animation [71], have been used in computer animation [38].

Finally, we can said that conventional animation is oriented towards the production of characters or 2D appearance images, and its great advantage is the flexibility obtained for being done by hand, with is also its great disadvantage, due to the big number of drawings that are necessary for producing a feature film.

1.4 Computer assisted animation

Now we can see how the computer is used for assisting the production of animation. Though it is very useful, the more time-consuming tasks, to draw key frames and inbetweening, can not be done fully automatically.

Following the production phases, the computer can be used in:

1. Drawings creation
   (a) Key frames can be digitalized.
   (b) Key frames can be created with a graphic editor.
   (c) Complex objects can be obtained with programs.
2. Movement creation
   Currently, inbetweening can not always be done, or it has not the needed flexibility.
3. Coloring
4. Filming
5. Postproduction

Some developed systems are [41, 19].

The inbetweening will be detailed commented in (Page 10).

1.5 Computer animation

The computer is the needed tool in computer animation. The big difference in relation with classical animation and computer assisted animation is that, once the models and other data are given, the computer controls the production of images, in the sense that there is not human indication. This characteristic joined to the possibility of working with high amounts of data with speed and precision, turns the computer the tool that allows this kind of animation.

The great advantage of computers is that can produce virtual worlds.

The phases of a computer animation, given the history, are:

1. Modelling
2. Animation
3. Visualization

1.5.1 Modelling

The most used modelling scheme is based in surfaces. There are three possibilities:

- Polygons
  This type of model is very used from the beginning of computer graphics. There are many developed techniques, z-buffer, shading (Gouraud, Phong [21, Pages 598-599,738-739]), and so on.

- Algebraic or implicit surfaces
  Some of these surfaces are quadrics: spheres, cylinders, cones, and ellipsoids. A new set are the superquadrics [3].

- Parametric surfaces
  The kind of surfaces that can be used are:

  - Bézier
  - Cubic Splines
  - B-splines
  - Coons
  - Gordons

1.5.2 Objects creation

The most used methods for creating objects are:

- Digitalization of 3D objects
- 2D digitalization and 3D reconstruction
- Using procedures

1.5.3 Animation

New techniques have been developed for computer animation. The classical techniques are also used. In Figure 1.10 is shown the squash and stretch technique, and in Figure 1.11, there is an attractive design.

Generally, the figures are articulated, composed of several pieces joined in a hierarchical structure. This animation is in most cases of parametric type. There is also inbetween process in computer animation.

Dynamic and cinematic are used for obtaining more realistic images. There is a lot of paper related with this theme [2, 6, 10, 32, 51, 55, 70].

There have been developed specific techniques for resolving special problems, as for example, face animation [72, 33, 77] , cloth objects [8], hair [14, 35], gases [16, 68, 69, 73], plants [56], and so on.

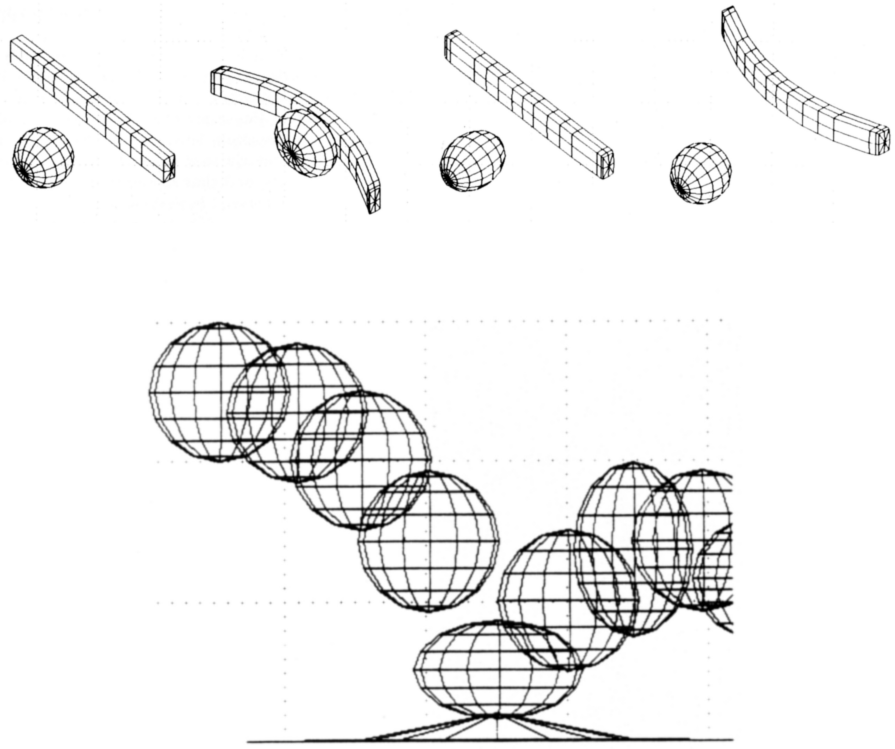

Figure 1.10: Squashing and stretching in 3D [55].

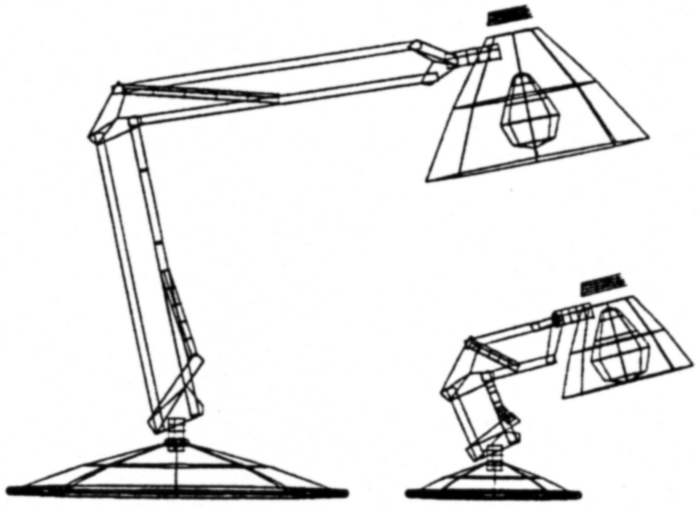

Figure 1.11: Attractive design in 3D [38].

1.6 Inbetween

The inbetweening consist in producing the intermediate states, given one initial and one final state. The meaning of state depends upon the kind of data that is used. If the data are images, we say that it is key frame inbetween. If the data are parameters

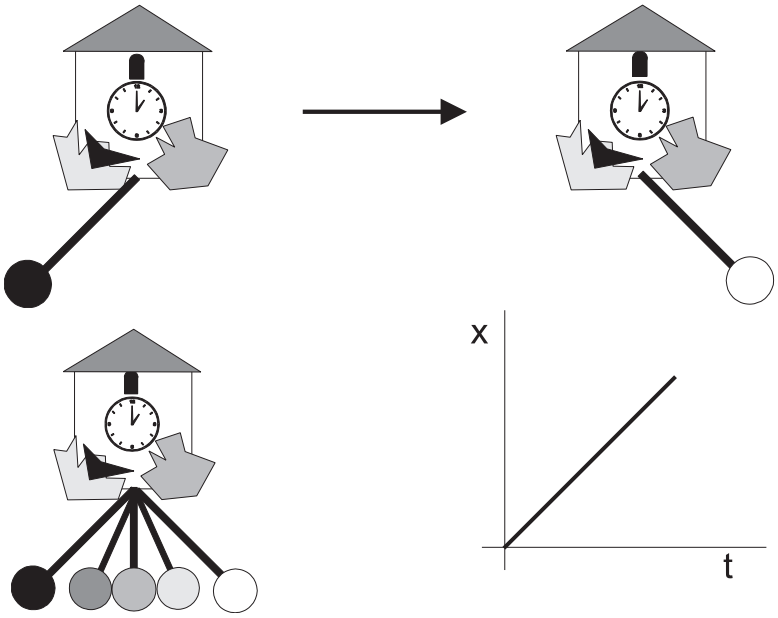

Figure 1.12: The linear interpolation of images can produce wrong results.

we say that it is parametric inbetween.

- Key frame inbetween
  This is the kind of inbetweening that is used in classical animation. Though it is very easily done by humans, it is very difficult for computers. The human inbetweener can infer 3D information lost in projections [9] from the 2D images, but this process is very difficult for a computer.

  Even though there is an exact correspondence between the initial and final images, it is possible that the result is wrong (Figure 1.12).

- Parametric inbetween
  This type of inbetween is normally used in assisted computer animation and computer animation. Instead of obtaining images, intermediate values between one initial and a final values are obtained. Then they are used for producing images. In the pendulum example, the parameter is the angle, which is interpolated between two extreme values (Figure 1.13). Although there a correct

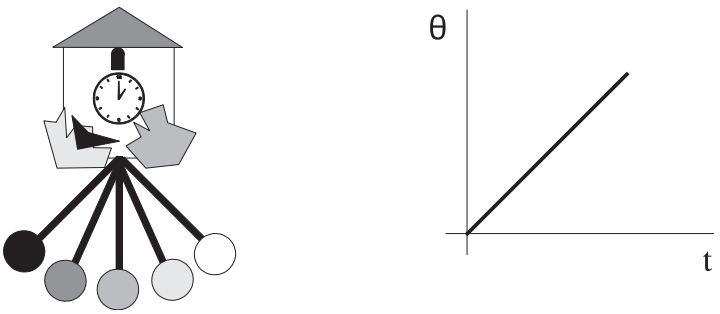

Figure 1.13: Linear parametric interpolation.

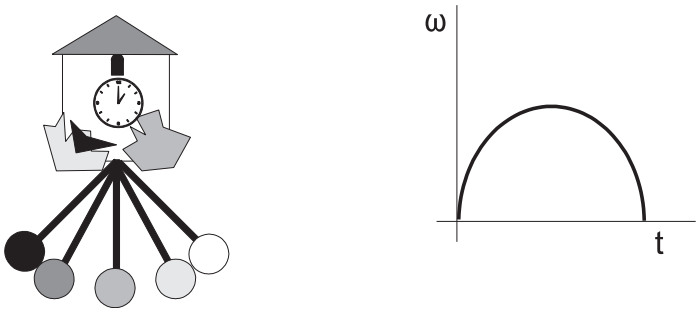

Figure 1.14: Parametric interpolation with physics law.

visual appearance, due to the linear interpolation, the obtained movement is not real. The function used for doing the interpolation can be of any type. Depending upon whether the interpolation function uses or not a physics law, some authors [44] divide this type of interpolation in key frame parametric animation (without physics law) and algorithm animation (with physics low). In Figure 1.14 can be seen the result of applying a physics model to the same example.

1.6.1 Methods of interpolation

The interpolation can be or can not be linear, using images or parameters.

Different solutions are:

- Linear interpolation

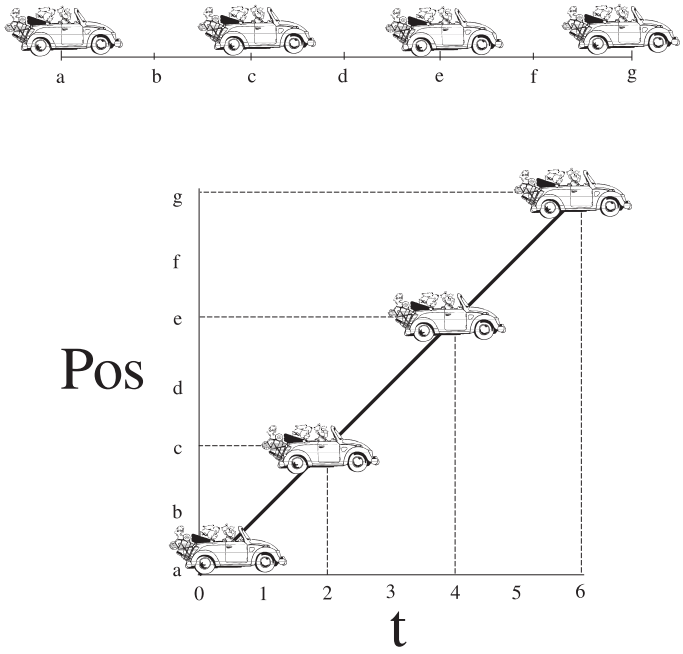

Figure 1.15: Linear interpolation method.

The used function is a straight line. We want that the car is in position A in time $t = 0$ and in position G in time $t = 6$. The result is shown in Figure 1.15. No, we want to impose the next conditions:

Time	Position
0	A
2	B
4	F
6	G

We can use a linear-by-sections interpolation. The problem is that in union points there are discontinuities which represent changes in velocity (Figure 1.16).

- Non-linear interpolation
  The solution is in using non linear functions for interpolating. In Figure 1.17 there is a possible solution for the previous problem.

  Some laws normally used in animation are:

  1. Law=$K * frac$ const
  2. Law=$(1 - (cos\theta * frac/2))$ aceleration
  3. Law=$sin(\theta * frac/2)$ desaceleration
  4. Law=$(1 - cos(\theta * frac))/2$ aceleration-desaceleration

- P-curves
  Another possibility is the P-curves method (R. Baecker (1969)).

- Moving Points Constraints

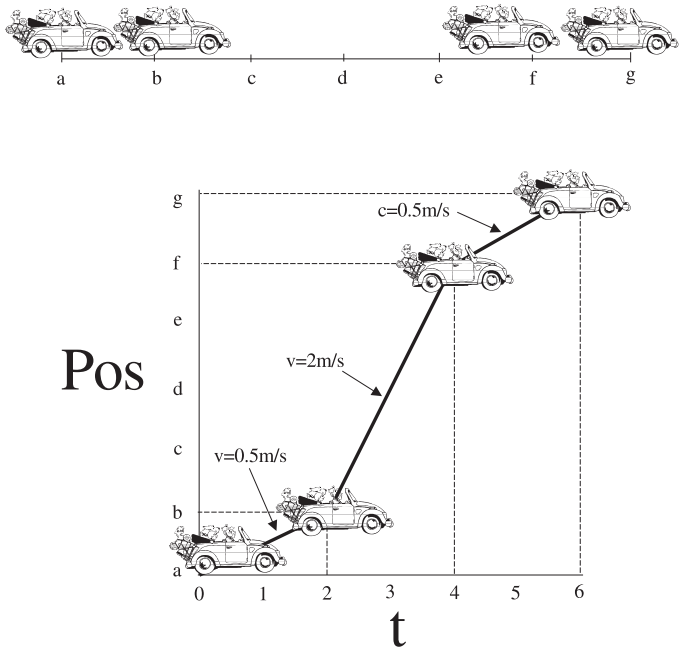

Figure 1.16: Linear by sections interpolation method.

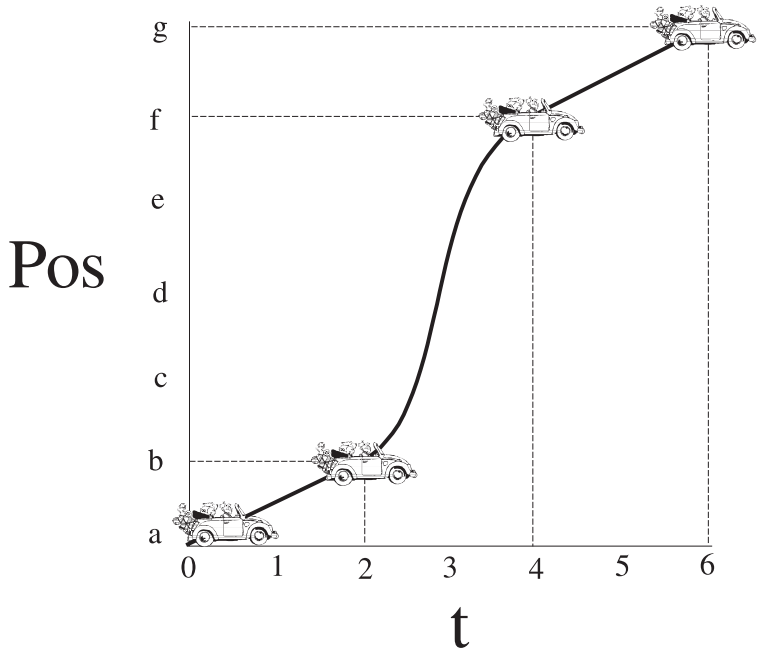

Figure 1.17: Interpolation using a cubic spline.

1.6.2 Skeleton technic

One solution for resolving the correspondence problem between images is the skeleton technique [7]. The skeleton is a simplification of the object, based in a simple polygons mesh. In Figure 1.18 there is an example.

1.7 Objectives: characteristics of an ideal system

Once the characteristics of conventional, assisted, and computer animation are presented, the advantages and disadvantages can be shown:

- Conventional animation
  It has a particular appearance, is done by hand produced, and is oriented to produce tooned characters. It is very flexible but costly.
- Computer animation
  It is oriented to produce very realistic 3D worlds, though in last years there are also films that have 3D tooned characters (Toy Story, Bugs, Antz).

The advantages and disadvantages of both types of animation are complementary. So, the ideal system will have the next capabilities:

- Producing animation with 2D and 3D aesthetic.
- Great flexibility in creating and transforming actors which can be characters, and real and invented objects
- Economic production due to the automatization.

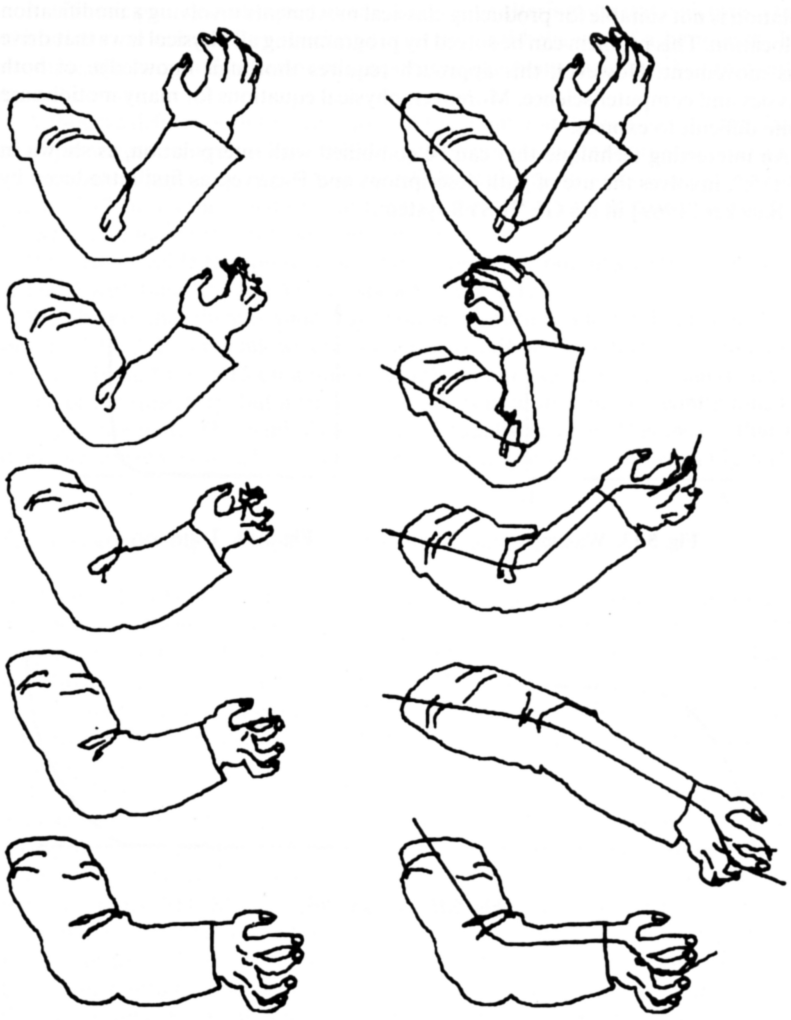

Figure 1.18: Deformation example using skeleton technique [7].

The ideal system can not be reached starting with a 2D system, due to the problem previously commented. A 3D system has almost all the necessary capabilities but the look of images.

So, we need a 3D system that can "toonize" the images.

The elements that are characteristic of classical animation drawings are shape lines and flat color. When drawings are oriented to illustration shape lines aesthetic can be even more important.

Shape lines are the lines that define the form of an object (Figure 1.19). We can distinguish between the lines that represent the physical limit of the object, the silhouettes, and the rest of lines that limit other visual characteristics. We call external sketches to the first kind of lines, silhouettes, and internal sketches to the second one. The possibility of obtaining more than one silhouette only occurs in concave objects (Figure 1.20). Normally, internal sketches represent change in color or shade.

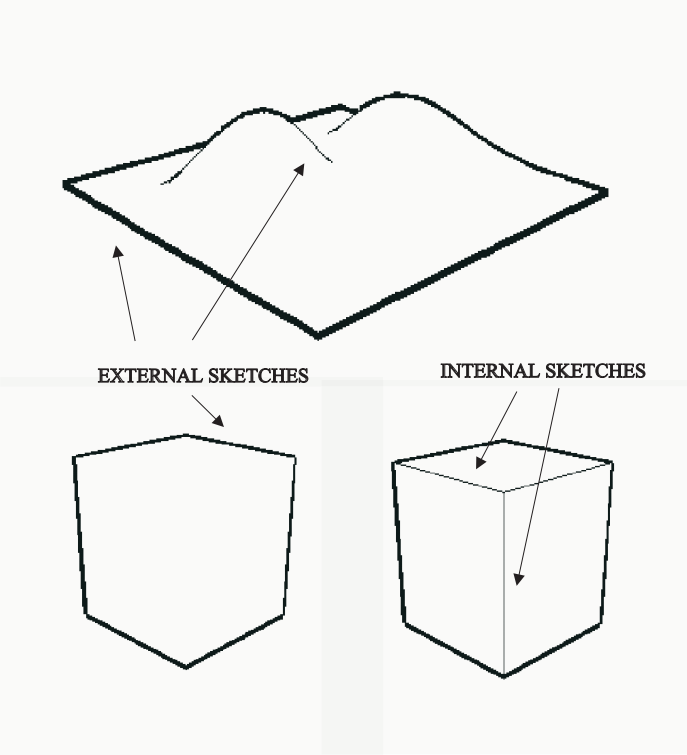

Figure 1.19: Shape lines (external and internal sketches).

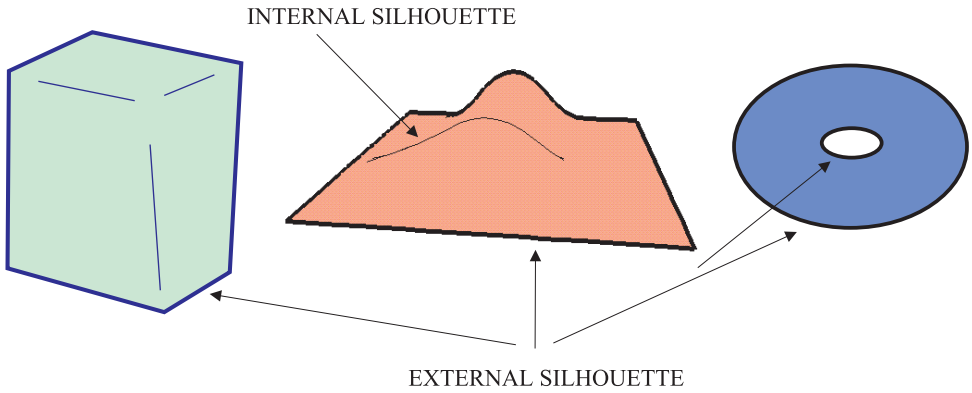

Figure 1.20: External and internal silhouettes: external sketches.

The other component of animation is animation itself, the production of movement. The method must be very expressive.

So, we have to resolve two main problems: the visualization problem and the animation problem.

1.8 Previous works

In the last few years there has been a great interest in which is called non photo realistic rendering, and pictorial or expressive visualization [11, 15, 23, 24, 26, 25,

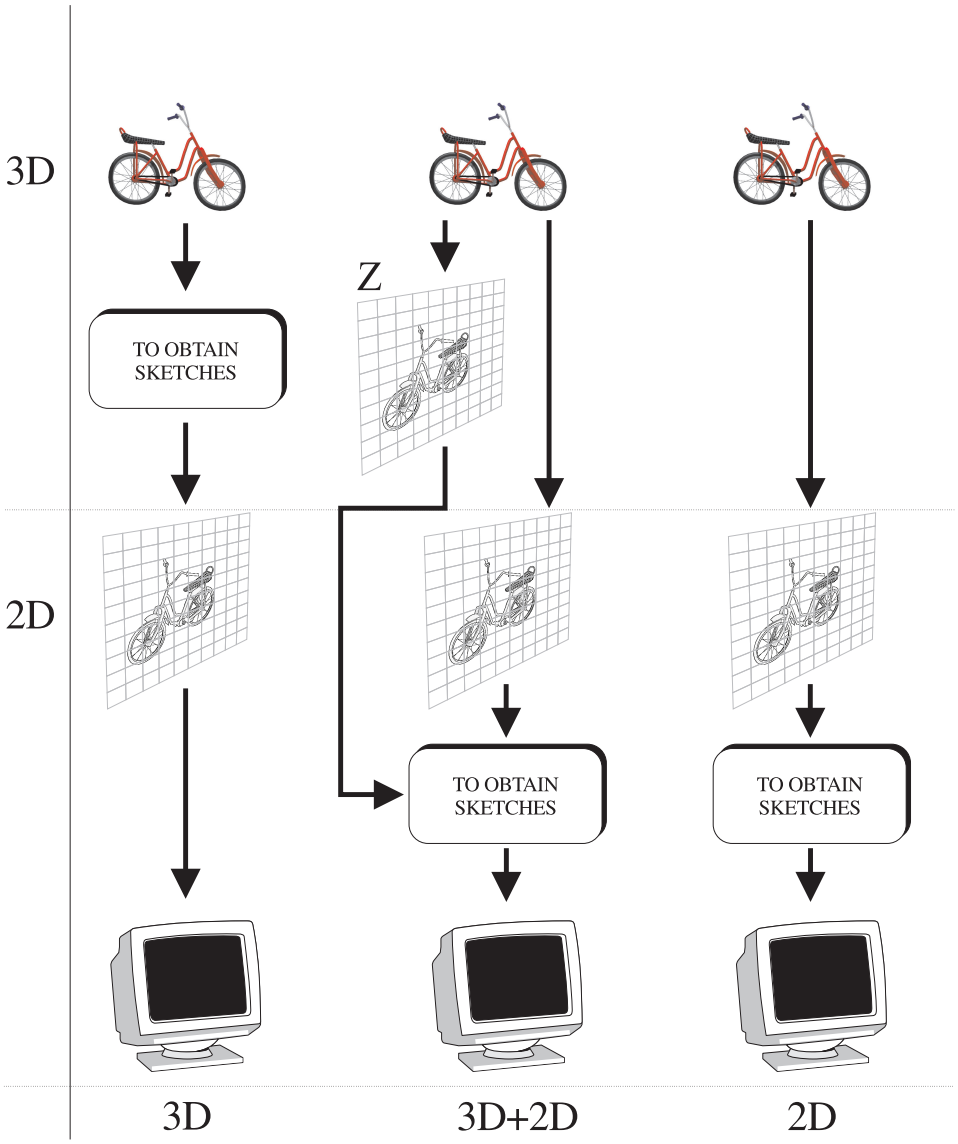

Figure 1.21: Classification of algorithms in relation with the used space.

27, 28, 30, 37, 40, 45, 46, 47, 48, 49, 60, 61, 62, 67, 75, 76, 80].

1.8.1 Visualization

There are several works about methods for obtaining silhouettes, contour lines, textures sketches, and so on. A revision of expressive visualization can be seen in Lansdown [37].

In this section, there is a revision of works that allow to obtain silhouettes from a 3D model. We do not consider methods based in filters.

There is a common element that allow us to classify the methods: the dimension of the space used for obtaining the silhouettes [46]. The first category methods, 3D, uses the 3D model and obtain 3D information. The second category, 3D+2D,

uses a 3D model from which extract some information, saved in 2D format, which is used in second place for obtaining the silhouettes. The final category, 2D, uses the 2D information saved in the frame buffer or other buffers (stencil, and so on). The classification can be seen graphically in Figure 1.21.

General characteristics

The algorithms that use polygonal models form the silhouettes with edges. If we have well formed closed 2-manifold objects, every edge is shared by two faces. If we have the observer in the positive part of z axis in a normalized coordinate system, the edge that has one face visible and the other invisible will be part of the silhouette. In open surfaces, edges that have only one face must also be added.

The models based in parametric surfaces have a continuous definition. Given a object defined by a parametric surface $S(u, v)$, the function that defines the normal vector for every point can be obtained, by using the tangent function for the u and v directions $T_u(u, v) = \frac{\partial S(u,v)}{\partial u}$ and $T_v(u, v) = \frac{\partial S(u,v)}{\partial v}$. The normal vector is obtained using the dot product of both equations; $T(u, v) = T_u(u, v) \otimes T_v(u, v)$. A problem is the apparition of point with not normal vector [18, Pages 281-284]. If there is not such a problem, the function $T(u, v)$ is a five degree equation in u and v, though it can be simplified to degree 3 [52]. With this function, the silhouette is defined by the next condition $T_z(u, v) = 0$.

A original scheme of silhouette extraction is in [52, Pages 533-538].

- **3D Methods**

  - Flat polygons
    The first algorithm that allows to obtain silhouettes was Roberts's one [21, Pages 665-666]. A more sophisticated algorithm was presented by Appel [21, Pages 666-667]. Markosian [45] presents a new algorithm based on Appel's method.
  - Non-polygonal surfaces
    Elber y Cohen propose a method that works with parametric (splines) surfaces [17] (Figure 1.22).

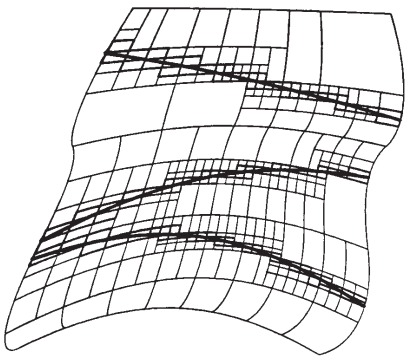

Figure 1.22: Extraction of silhouettes in parametric surfaces [17].

Sederberg y Zundel present a method for visualizing algebraic surfaces using a scan line algorithm [66].

A	B	C
D	X	E
F	G	H

Figure 1.23: Pixels position for applying Sobel's operator.

- **3D+2D methods**
  One important method is the work developed by Saito y Takahashi [59] with their G-buffers. They use the Sobel's operator for obtaining the silhouettes from the 2D save information ($g = \frac{|A+2B+C-F-2G-H|+|C+2E+H-A-2D-F|}{8}$) (Figure 1.23). In Figure 1.24 there is an example of its application.

  Salisbury [60, 61, 62], as well as Winkenbach and Salesin [75, 76] have developed several methods oriented towards the illustration (Figure 1.25). In [75, 76] they

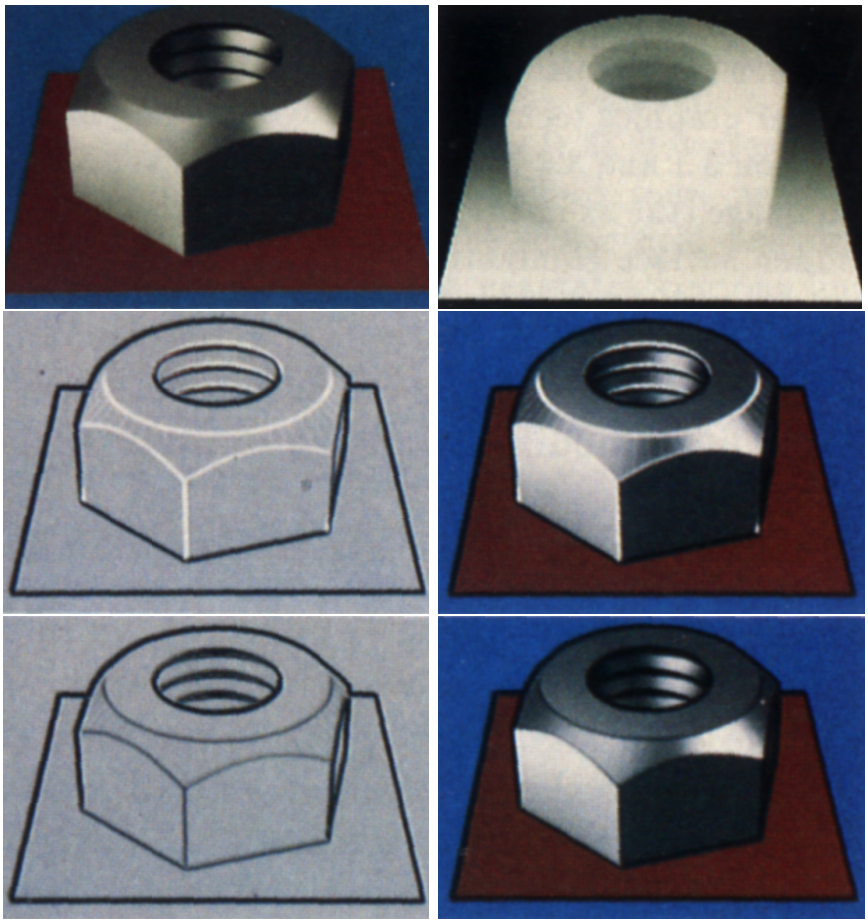

Figure 1.24: Example of G-buffers application [59].

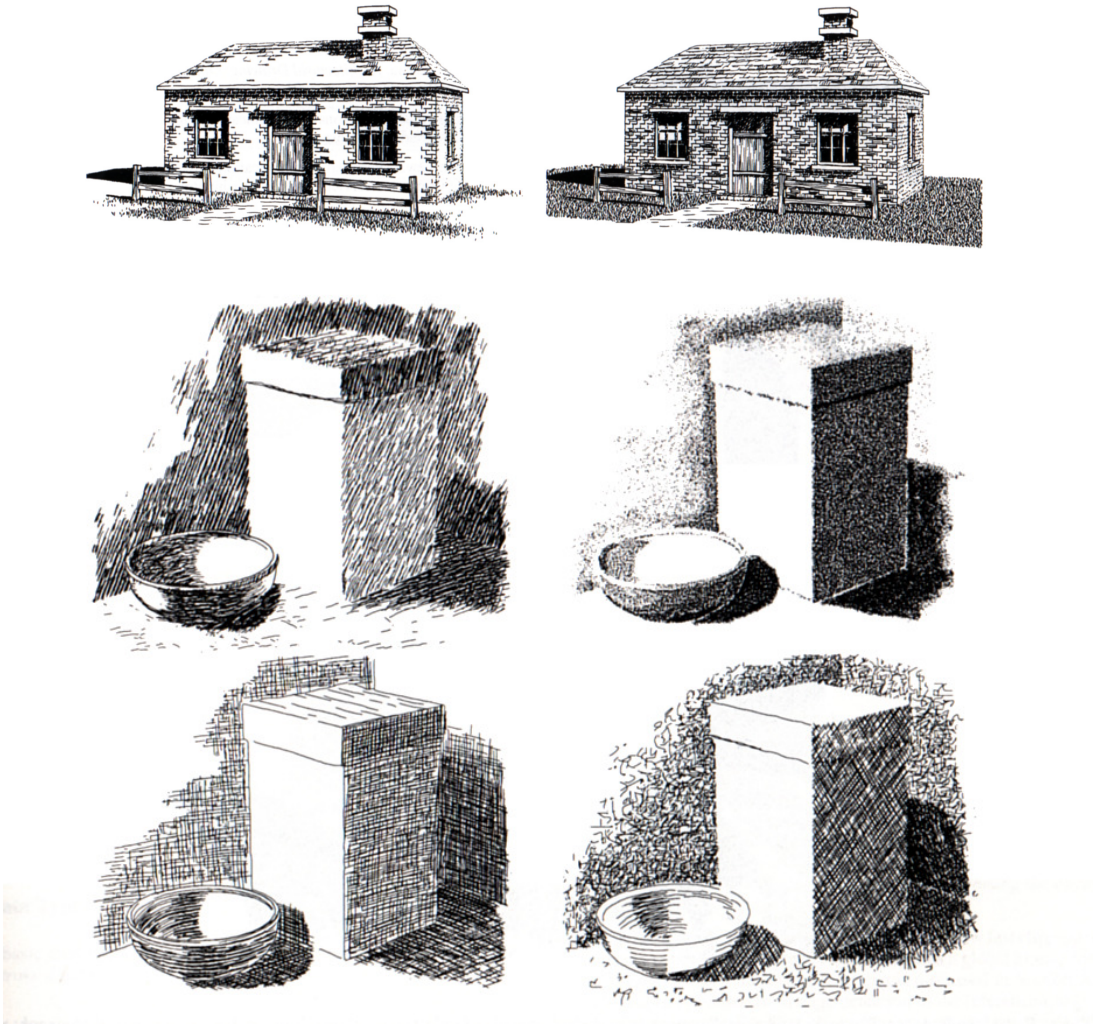

Figure 1.25: Illustration examples [75, 61].

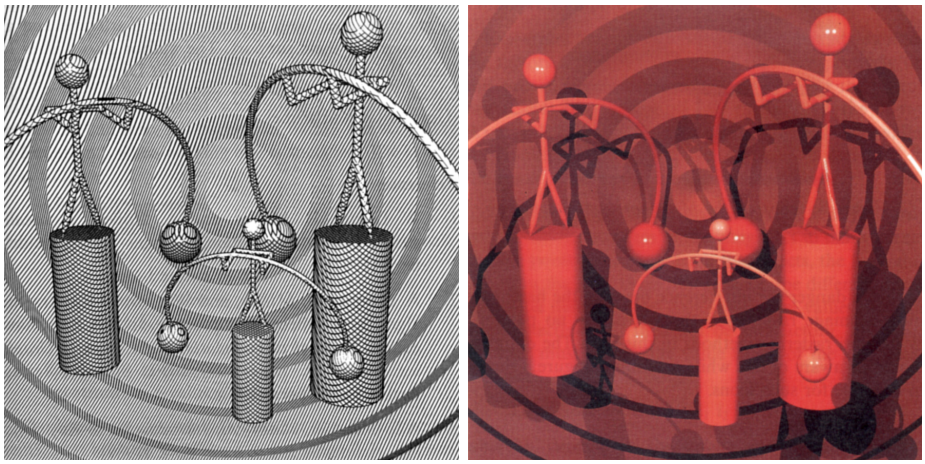

Figure 1.26: Illustration example [40].

use what they call texturized sketches. Another 3D+2D method is the Leister one [40], which uses ray trancing for obtaining the information (Figure 1.26).

Richen and Schofield [57] present another algorithm that uses the material buffer, in a la Saito manner.

- **2D methods**
  P. Rustagi [58] developed a method that works only with 2D information (Figure 1.27).

An interesting work for its aesthetic results is developed by Hsu [30] (Figure 1.28).

1.8.2 Animation

The most common schemes used for producing deformations are the Free Form Deformation and the Non-Linear Transformation. These methods have the advantage of being representation independents

- **Free Form Deformation**
  The method was developed by [65]. This technique has also been used by [7], and Litvinowickz [41] uses it with his 2D animation system. Coquillart [12] presents an extension that allows a variety of meshes, and then [13] extend it for being used as an animation tool. A new extension is proposed by [43]. Hsu [30] uses the FFD for obtaining what he calls skeletal strokes (Figure 1.28). In [31] is presented a direct manipulation method for FFD's. Karan [34] use control curves in a way similar to the NLT.
- **Non-Linear Transformation**)
  This method was developed by Barr [1]. Because this is the method that we have implemented, it will be explained with mode detail in next chapter (Page 31).

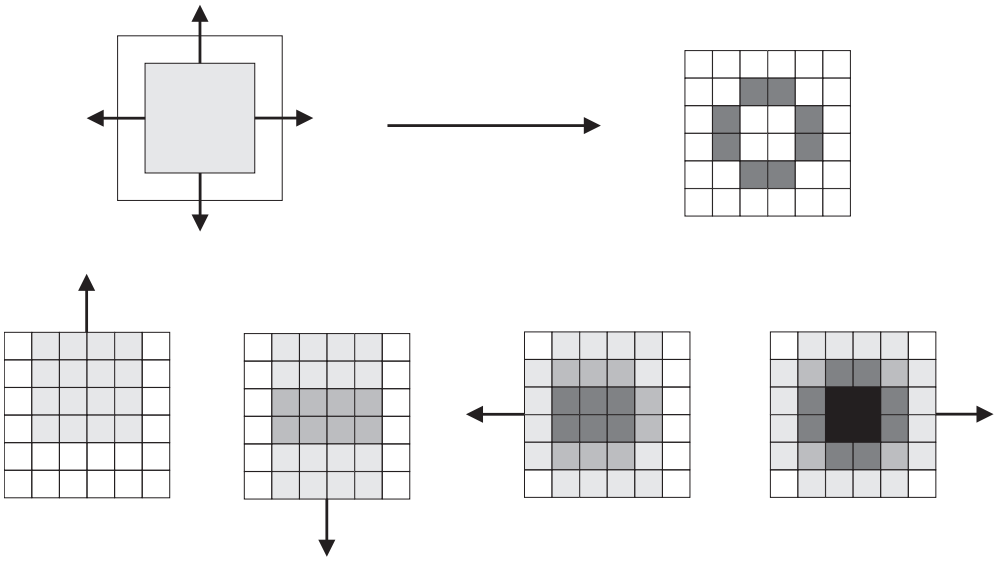

Figure 1.27: Silhouette obtention with Rustagi method.

Figure 1.28: Example of several kind of lines using skeletal strokes [30].

Another possibility is to use implicit surfaces. The first work is Blinn's one [5] (Blobby molecules), then Nishimura et al. (Metaballs) [54], Wyvill et al [78, 79] (Soft objets), and others [53] (Figure 1.29).

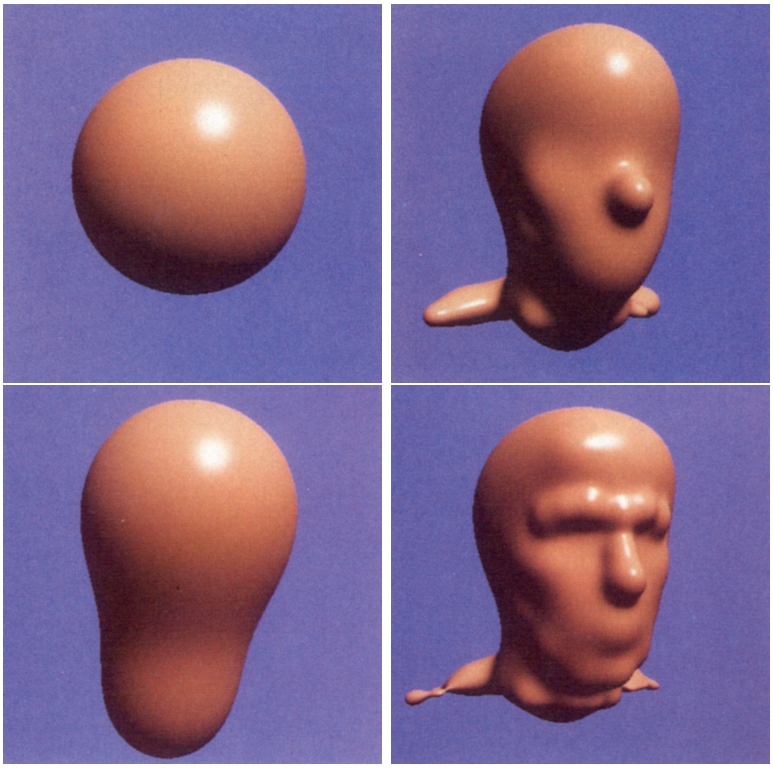

Figure 1.29: Example of soft objects [53].

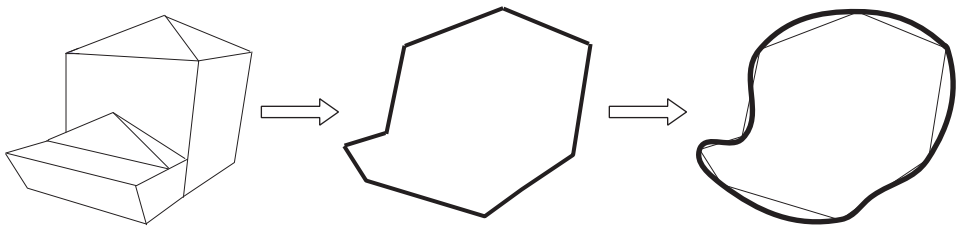

Figure 2.1: Obtention of sketches.

Chapter 2

Model

The objective of this thesis is to propose a model that allows us to obtain animation with a classical look from a 3D model. The conditions have been exposed in the previous chapter. The components of the model that we have developed, called *Alhambra*, are shown in this chapter.

Once the objectives were defined, several methods were developed, that are commented in the previous developments section. These methods helped us to define the goals that we need to find. The main problem is divided in two: a visualization problem and an animation one. The solution to the first problem is found using sketches and *Virtual Lights* model. The solution of the animation is found in an extension of the Non-Linear Transformation, as an alternative method to Free Form Deformation.

2.1 Previous developments

There were three previous developments. The first one used a polygonal model from which some information was extracted to obtain the sketches. The second development used the image space for obtaining the sketches. The last one is related with the computing of intersections.

Though in some cases they were not good solutions, they provided the main ideas for developing the final method.

The first method [50] was oriented to the production of sketches. A very simple polygonal model was used. Cubes, pyramids, spheres were used as basic construction

elements. First, the vertices that form the silhouette were marked, then they were projected. Sketches were constructed using B-splines that passed through those projected vertices (Figure 2.1).

The shapes that we could obtain have some limitations due to the use of the simple model. They could not be observed from any place (Figure 2.2). Also, when some movements were done, specially rotations, the coherence of the shape was lost. The solution was to improve the resolution of the model, but this removed the fist advantage: its simplicity.

There were also problems with filling the figures, because the B-spline added some space that did not exist in the model. The problem appeared when there were several objects. The main ideas extracted from this development were: in case of using a polygonal model, it must be enough complex to represent correctly the shape of the object. The second idea was that the 2D space have "to summarize" the information of a high level space. The recovery of the lost information in the projection can produce incorrect results, so, at least, the model must be three dimensional.

The second method that was developed obtained sketches from information that was saved in a frame buffer-like data structure. It was a image space method. The main idea was that in a polygonal model, when back face culling is achieved and faces are drawn, the edges that form sketches are traversed only one time, whilst

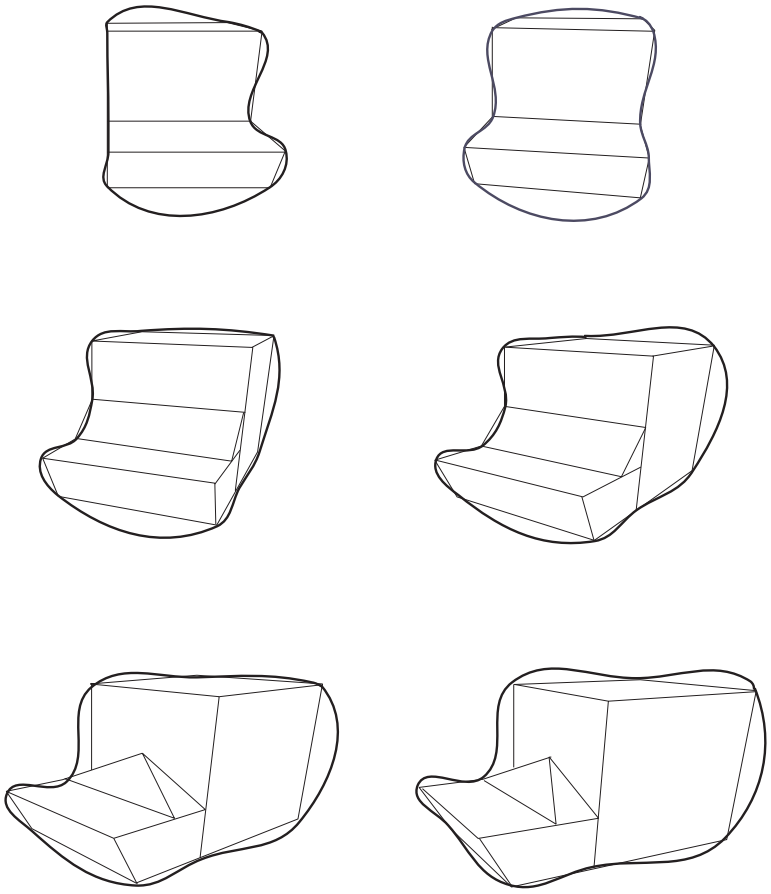

Figure 2.2: Model with different phases of refinement.

others edges are traversed two times. The saved information was about how many times a vertex was traversed.

The method was implemented and it rans but with low efficiency.

The third method that was studied was about computing the intersections between the silhouettes, because a concave object can has several of them (see the mountain in Figure 1.20). The operations were achieved in image space which produced problems of aliasing.

All obtained results, goods and bads, were used to define the right way to obtain our goals. The solution that we present is based in three elements:

- A 3D geometric model
- Sketches, defined from *"Virtual Lights"*
- The Hierarchical Extended Non-Linear Transformations model

These elements are more detailed commented in next sections.

2.2 3D geometric model

The first element is the geometric model. It must possess some special characteristics that permit to develop the others methods and techniques. The properties that must have are that its surface can be easily evaluated, that the normal vector can be evaluated in every point, that can be transformed in a intuitive and easy way, and must allow us to obtain the sketches. The selected model is a polygonal model based in triangles.

2.3 Sketches

The more important elements that produce a 2D-like drawing are the shape lines, joined to flat colouring (Figure 2.3). The shape lines are the more important pictorial elements of drawing and illustration. We call them sketches (Figures 1.19 y 2.3). It can be distinguished between the shape lines that represent silhouettes, interior and exterior, that we call external sketches, and shape lines that allow us to distinguish between several visual characteristics that we have called internal sketches.

Both types of sketches can be defined in an informal way as:

- External sketches
  External sketches are silhouettes, external and internal, that an object has when it is observed from a position. Silhouettes can be defined as the limit of visible and invisible parts of the object.
- Internal sketches
  Internal sketches are those shape lines that are not silhouettes. In general, internal sketches distinguish between parts of the object that have different visual attributes, for example, color or shade.

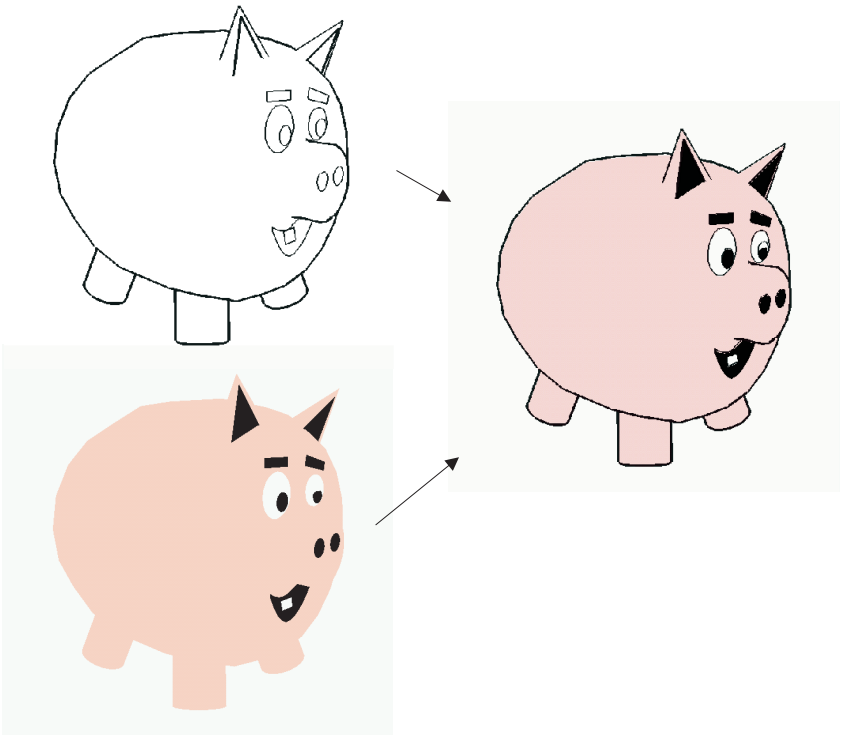

Figure 2.3: Shape lines (sketches) and flat colors.

2.3.1 Visualization algorithm

Now that we have an informal description of sketches, we can present the algorithm of generation of sketches:

for all Object **do**
To obtain geometric elements that form internal and external sketches
To compute the intersections between geometric elements
To form chains of geometric elements
To visualize the chains
end for

A geometric element is a part of a sketch, and so, its type depends on the selected geometric model. It can be an edge for a polygonal model or a curve for a model based on parametric surfaces.

These geometric elements will constitute sketches, so they must be selected depending on their visual attributes.

The *"Virtual Lights"* model is used for treating and obtaining the sketches. Once the geometric elements are selected the computing of intersections between them must be computed. This situation occurs with concave objects (the mountain in Figure 1.20). In such case, we need to determine what parts are visible and what are not. This and other parameters can be used for obtaining a great flexibility in visualizing sketches.

The geometric elements have to be joined to constitute chains. A chain is considered as an unique element with its own attributes. The chains can be open or closed ones. Sketches can be displayed with this information.

Now we can show a more formal approximation of all these elements.

2.3.2 *Virtual Lights*

While external sketches are easily define, it is not the same with internal sketches, because the first ones are fixed by the scene conditions while the second ones are more flexible, due to that they are defined by the user.

The *Virtual Light* model allows us to represent both types of sketches in the same way. And what is more important, it allows the user to define when, where, how and which geometric elements of the model will be selected as sketches.

Virtual Lights: selection and classification

The *Virtual Lights* model is composed of several elements:

- Virtual lights.
- Illumination models.
- A 3D geometric model.

Definition 1 *A virtual light is determined by a direction $\vec{v}$, or a position (x, y, z).*

In the first case, the virtual light is localized at infinity, we call it *non-local*. In the second case, the light is near the scene, and we call it *local*. Virtual lights are different of normal lights, in the sense that they do not contribute to light the scene. Also, every object has is own set of virtual lights. They have no intensity.

Virtual lights are used in conjunction with two illumination models: diffuse and specular.

Definition 2 *A diffuse illumination model associates a real value I between -1 and 1, to every point of the object, to each virtual light, with the expression:*

$$I = cos\theta$$

where θ is the angle that form $\vec{N}$, the normal vector of the surface and $\vec{L}$, the direction vector of the virtual light. This vector is equal to the direction vector for a non-local virtual light, and is computed as the vector between the position of the virtual light and the point for a local virtual light (Figure 2.4).

If $\vec{N}$ and $\vec{L}$ are normalized, the model can be represented as:

$$I = \vec{N} \cdot \vec{L}$$

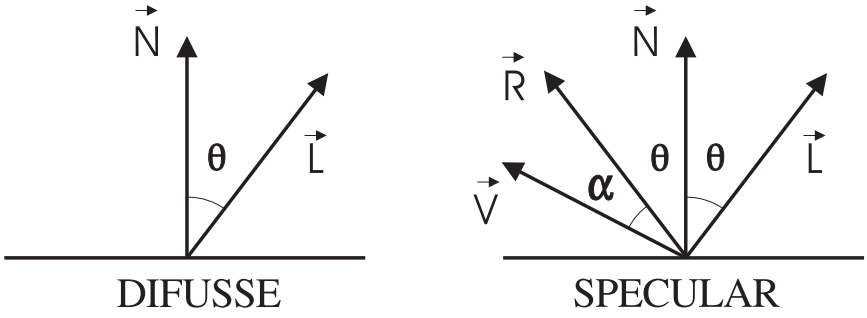

Figure 2.4: Parameters used with virtual lights.

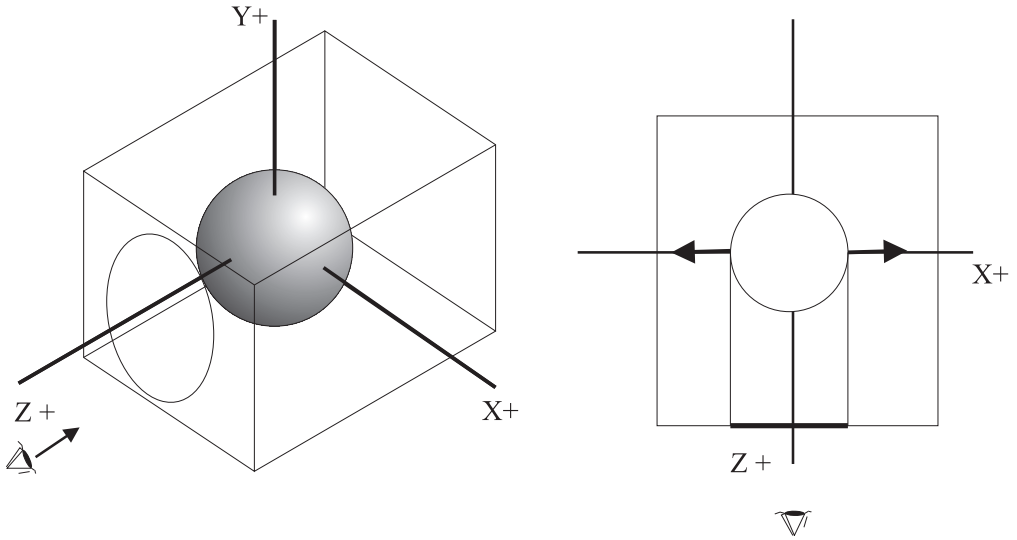

Figure 2.5: External sketches are object's silhouettes.

Definition 3 *A specular illumination model associates a real value I between -1 and 1, to every point of the object, to each virtual light, with the expression:*

$$I = cos^n \alpha$$

where α is the angle that forms $\vec{R}$, the symmetric vector to $\vec{L}$ in relation to $\vec{N}$, with $\vec{V}$, the observer's direction vector. The $\vec{L}$ vector is computed in the same way as previously commented for the diffuse model. The $\vec{V}$ vector is computed as the difference between the observer's position and the point. The n value represents the brightness of the object. (Figure 2.4).

As every virtual light has associated a illumination model, we have diffuse virtual lights and specular virtual lights. Due to our interest in changes of visibility or color, the intensity of the light and the reflexion coefficients can be simplified. The main difference between both types of virtual lights is that diffuse ones are independent of the observer's position, while with a specular virtual light sketches are selected depending on the observer's position, the object's orientation and the position of the virtual light.

Now we are going to define sketches:

Definition 4 *Given a 3D geometric model and a virtual light, a geometric element is an external sketch if for every point of such geometric element, the dot product between the normal vector $\vec{N}$ in that point and the direction vector of the virtual light $\vec{L}$ is equal to 0.*

This definition can be graphically seen in Figure 2.5.

A diffuse virtual light located at the same position of the observer is used for obtaining the external sketches. For the internal sketches, they are selected depending on not only the orientation but also the computed reflection. They can have associate any type of illumination model.

Definition 5 *Given a 3D geometric model, a virtual light, and two real numbers IC and RC, an internal sketch is a geometric element that complies with the next conditions:*

(a) The angle formed between the normal vector and the virtual light is equal to IC.

(b) The reflection, obtained using the illumination model, is equal to RC.

The objects have a set of values for every defined virtual light. So, they have not only different types of virtual lights but also different conditions for each one. This makes the method very flexible in obtaining sketches where and when the user wants.

As every type of virtual light has associated an illumination model, in some occasions it is necessary to combine the effects of all the virtual lights of the object. This has been implemented using logical operations. For example, given two virtual lights, a geometric element will finally be a sketch if it is a sketch for the virtual light_1 **OR** the virtual light_2, or when it is a sketch for the virtual light_1 **AND** virtual light_2, etc.

2.3.3 Obtention of sketches

It is necessary to obtain the relationship between the position of the sketches due to the possibility that there are intersections between them. When there is an intersection, a visibility change occurs in this point: one part of the object hides other. There can also be external sketches that are totally included in the interior of the object. In illustration and classical animation, the "character" of the sketch depends on its position and its relation with other sketches: an external sketch is normally drawn wider than an internal one.

Once the geometric elements are selected they are joined to form chains, which are considered as unique elements with their own attributes. That is, when a chain is visualized it is possible to take into account the previous and next state to take one or the other depending on the context, as for example to make a line wider, etc.

2.4 Animation

Once the elements that allow us to obtain the visualization are defined, we need to define a method or scheme that allow us to produce an animation. This method must be:

- Representation independent.
- Easy of using.
- Intuitive.
- Simple representation.
- Expressive (to permit a great variety of deformations).

It is of special interest the last one because is normal in classical animation. Though the Free Form Deformation is generally used, as commented previously (Page 21), it has a costly representation. So, we have decided to use the Non-Linear Transformation, because it has all the advantages exposed, and the fault of local control will be reduced with our extension.

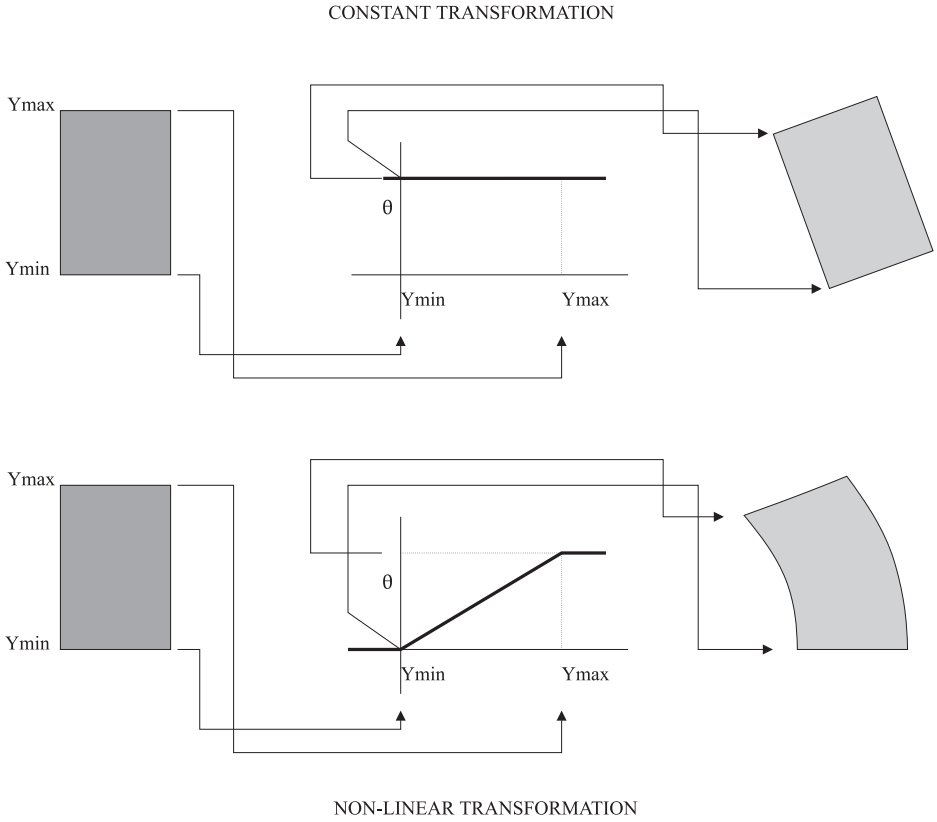

Figure 2.6: Constant and Non-Linear Transformations.

2.4.1 Extended Non-Linear Transformations

Non-Linear Transformations are a variation of geometric transformations; translation, rotations, scale and so on. In non-linear transformations, the transformation itself is changed depending upon the position. The function that relates position and transformation is called *control function*. A geometric transformation can be seen as a non-linear transformation, of which the control function is constant in the extension interval of the object (henceforth referred as constant transformations). More formally, given a transformation, T, which is constant for all vertices, a point (x, y, z) is transformed into $(x', y', z') = T(x, y, z)$. As can be seen in the example of Figure 2.6, the control function of the geometric transformation is constant whilst for the non-linear transformation the function is a straight line. A selection axis is chosen for every transformation. The selection axis points to which coordinate x, y, o z, will be the independent variable of the control function (Figure 2.7).

Barr defines the following NLT's:

- Tapering
  Tapering is a modification of scale (Figure 2.8, row 2).
- Twisting
  Twisting is a derivation of a rotation (Figure 2.8, row 3).
- Bending
  Bending through a selected axis is a composed transformation; in one region a bend is applied, and outside that zone a rotation and a translation are applied (Figure 2.8, row 4).

These transformations may produce some interesting results, but they are not flexible enough, so we present an extension that introduces more flexibility and the capability of deformation, as well as a more general treatment of transformation composition: Extended Non-Linear Transformations (ENLT).

Every geometric transformation can be used with ENLT's. A selection axis is also used. An *application* axis is introduced which gives more flexibility in deforming. It controls which coordinate , x, y or z, the transformation will be applied

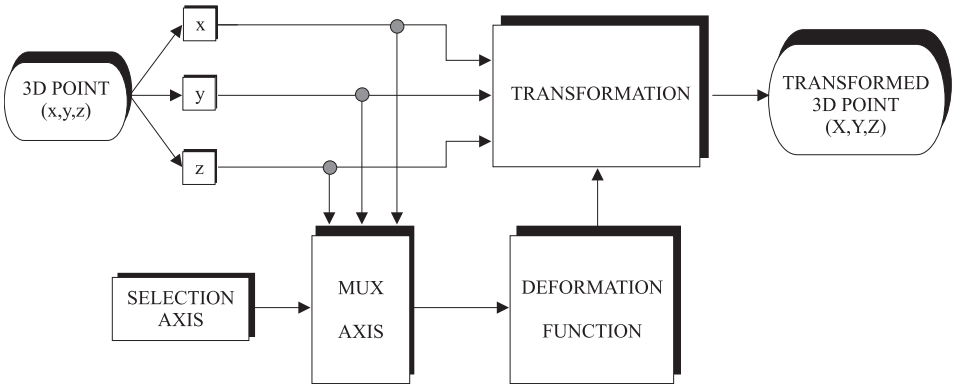

Figure 2.7: Flowchart of a Non-linear Transformation with selection axis.

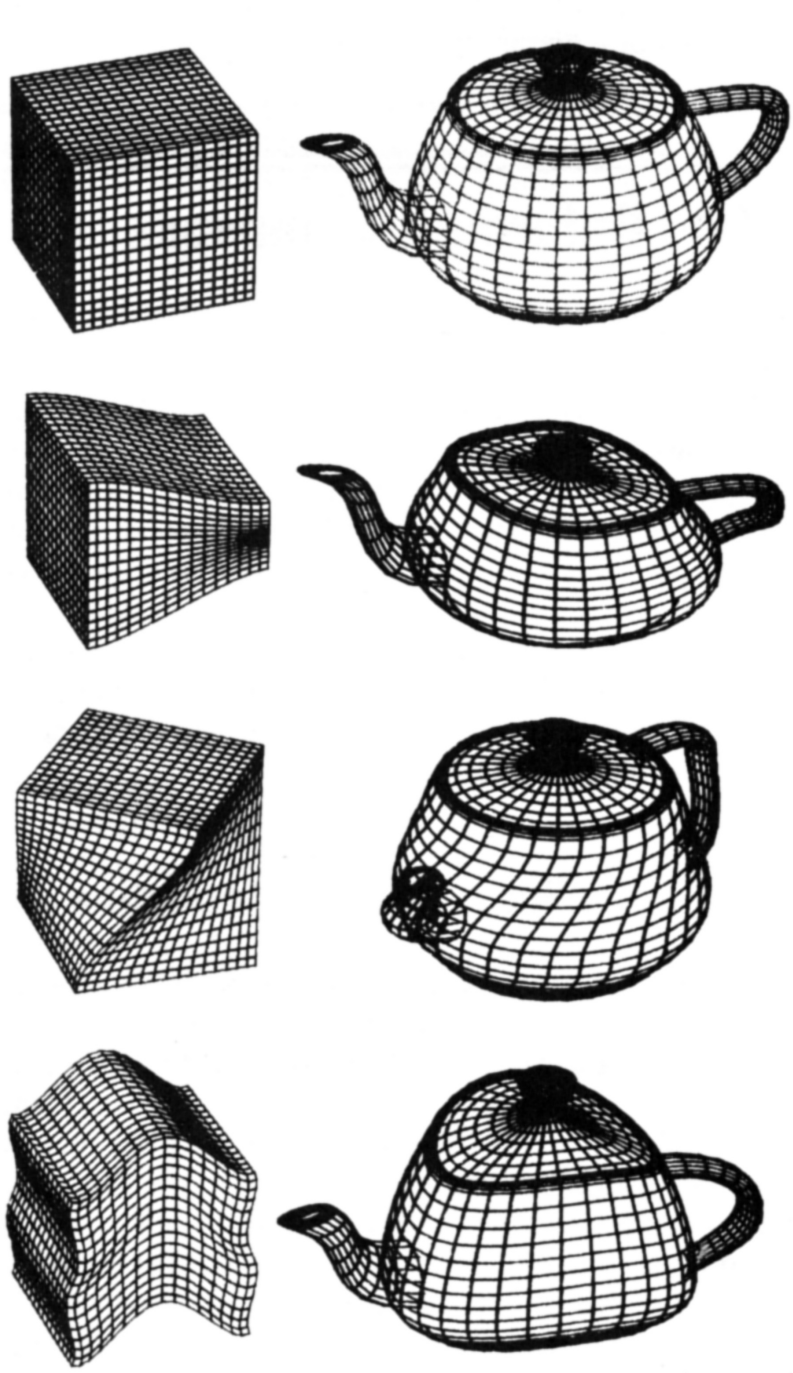

Figure 2.8: Barr's Non-Linear Transformations.

to (Figure 2.9). In Barr's transformations these coordinates are fixed. This is a generalisation of Barr's mechanism because any of Barr's transformations can be expressed as the composition of transformations, with the same selection axis but with different application axis. For example, a tapering with selection axis z will be $(x', y', z') = (rx, ry, z)$ with $r = F(z)$, while the proposed transformation will be:

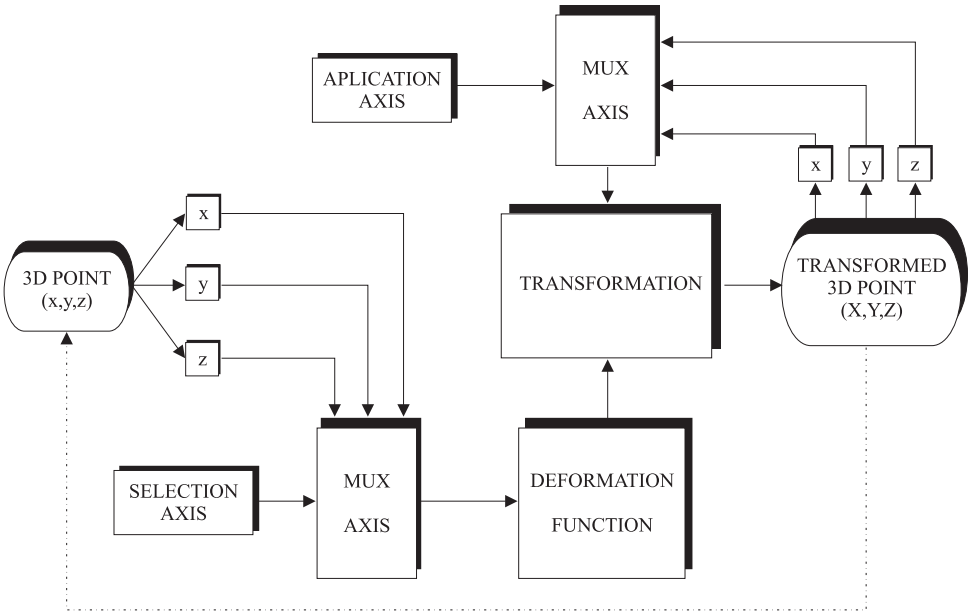

Figure 2.9: Flowchart of an Extended Non-Linear Transformation with selection axis and application axis.

Selection axis z

1) $(x', y', z') = (rx, y, z)$ $\quad r = f(z)$ Application axis x
2) $(x'', y'', z'') = (x', ry', z')$ $\quad r = f(z')$ Application axis y

The final result is the same, but with the inclusion of the application axis, we can produce transformations that are not possible with Barr's scheme.

A *control function* is a function that defines how the parameter that control the deformation changes. This function depends upon the value of a coordinate, which itself depends in turn on the selection axis (Figure 2.6). This function must be defined in an interval which comprises a minimum and a maximum value corresponding to the minimum and maximum values of the vertices for the co-ordinate indicated by the selection axis. The shape and limits of the control function are previously defined by the user, but the limits can be changed dynamically. This change allows us to define a zone where the user wants the transformation to be applied. Outside this region, the function can be defined in two ways; either by taking the constant values of the extremes or by allowing the function to be valid outside the extremes. One example of application of functions can be seen in Figure 2.6.

As can be seen, once the control function is defined, the values obtained from that function are related to the value of the coordinate, which depends in turn upon the selection axis. Once the vertices are transformed, the extreme values may be different, and so, the control function must take them into account. These changes are not known before the transformations are applied, so we need a mechanism that will be time independent. One possible solution is to normalize the control functions, which implies a normalization of the object's geometry. This obliges us to apply a normalization of the geometry every time a transformation is applied. But even this solution can produce non-intuitive results.

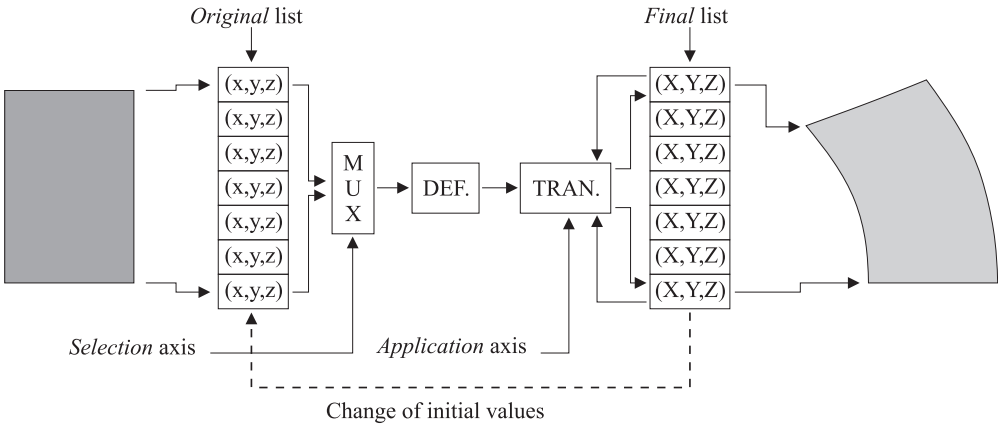

Figure 2.10: Method of initial list and final list.

The proposed solution is based on using the initial values of the object as values for defining the transformations. That is, the initial object, when it is not yet transformed, is used for applying the transformations. It is like a master object from which the transformed versions are obtained. This solution is implemented by means of two vertex lists: one list is called *initial* and the other list is called *final* (Figure 2.10). The initial values of the vertices are introduced into the initial list, which will be used for obtaining the values that will be used in the control function, depending upon the selection axis. The transformed vertices are in the final list (initially they are equal to the ones of the initial list). Any vertex of the final list corresponds to the transformed version of the vertex that occupies the same position in the initial list. This method allows us to apply successive transformations, with intuitive results. It also allows the apparition of the application axis. As commented above, the functions are defined depending upon minimum and maximum values, which can be adjusted dynamically by several control parameters. Thus, as long as the initial list does not change, these values do not change either.

2.4.2 Hierarchical Extended Non-Linear Transformations

Our version of Extended Non-Linear Transformations allows us to achieve interesting results, but there is a problem when the models are not simple, that is when they are composed of several parts which may involve their own transformations. Normally there is also a hierarchy which imposes the order in which the transformations must be applied. The problem with constant transformations is solved using a stack of transformations (PHIGS, OpenGL). There are two problems with ENLT's when they are combined with constant transformations. We propose Hierarchical Extended Non-Linear Transformations (HENLT) as another extension for resolving these difficulties.

The first problem consists in resolving the way the transformations that a child-object inherits from a father-object are applied, because the deformations are defined according to the initial coordinates of the objects. There are several possibilities concerning how the coordinates of the child-object are related to the definition of the control function of the father-object; the coordinates are totally included, partially included, or totally excluded (Figure 2.11).

The proposed solution is illustrated with a simple example which simulates a ”tooned" house (Figure 2.12). There is a hierarchy in which the roof, eyes, and mouth depend upon the house's body (Figure 2.13). The body movements must be transmitted to the rest of the elements, but on top of this every element can also have its own transformations defined. In Figure 2.12, the body of the house changes while the roof and the mouth follow the movement of the body without being transformed, and the eyes are pushed out.

The first problem is resolved by positioning the dependent object inside the range of the control function of the object upon which it depends. For example, the eyes must follow the house's movement. In this case, the definition range of the eyes is completely included within the houses's definition range. The constant transformations are used when an object has to be included in a hierarchical form within another object, changing the initial values to the transformed values, as explained above. In the example, the eyes must be translated to the correct position with a constant transformation, which will also change the coordinates of the initial list.

The second problem concerns the validity of the control function's range. For an object whose range is totally included, the solution has been explained in the example commented on above. It can happens that the range is partially or totally excluded. This is true of the roof, which is completely outside the house's range, but must follow its movements. The proposed method consists in controlling the extension of the control functions (Figure 2.14). If one object depends upon another, some attributes of the child-object will control the way in which the father's control function is extended.

The possibilities are:

(a) If the function is defined with constant values in the extremes, we compute the curve's slope at the extreme points and extend the function linearly.

(b) If the function is defined outside the limits, we expand these limits.

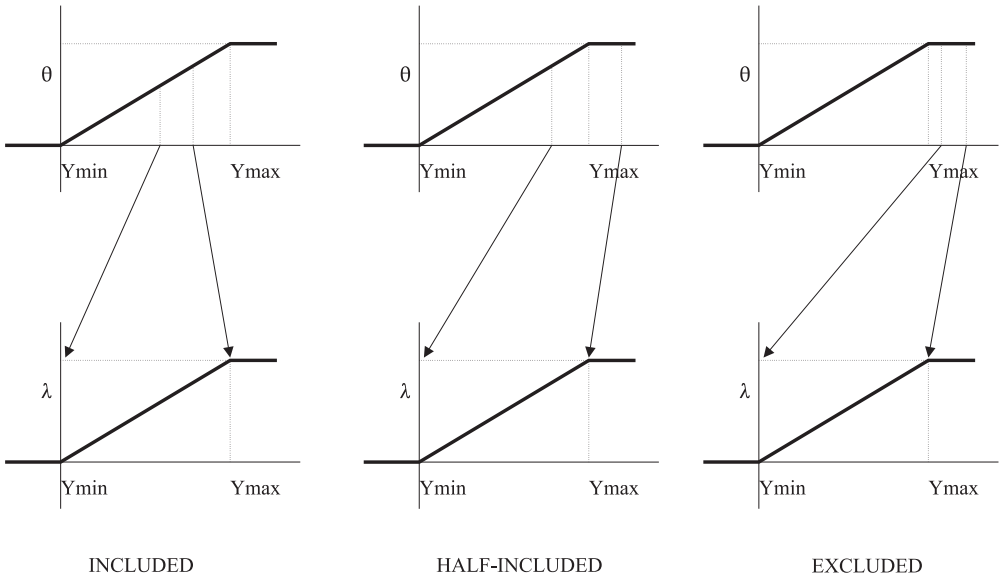

Figure 2.11: Possibilities of relation between control functions in a hierarchy.

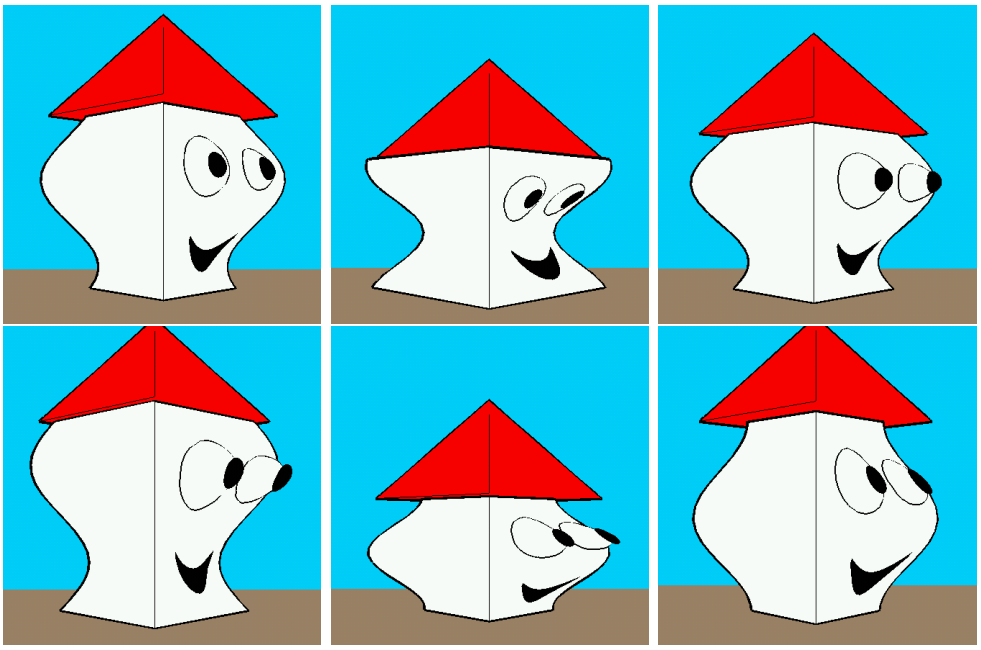

Figure 2.12: Example of Hierarchical Extended Non-Lineal Transformation.

This expansion operation can be applied individually with every transformation of the father-object, which allows more flexibility or else can be defined in a global way with less control. The hierarchies are defined by the user depending on his or her needs.

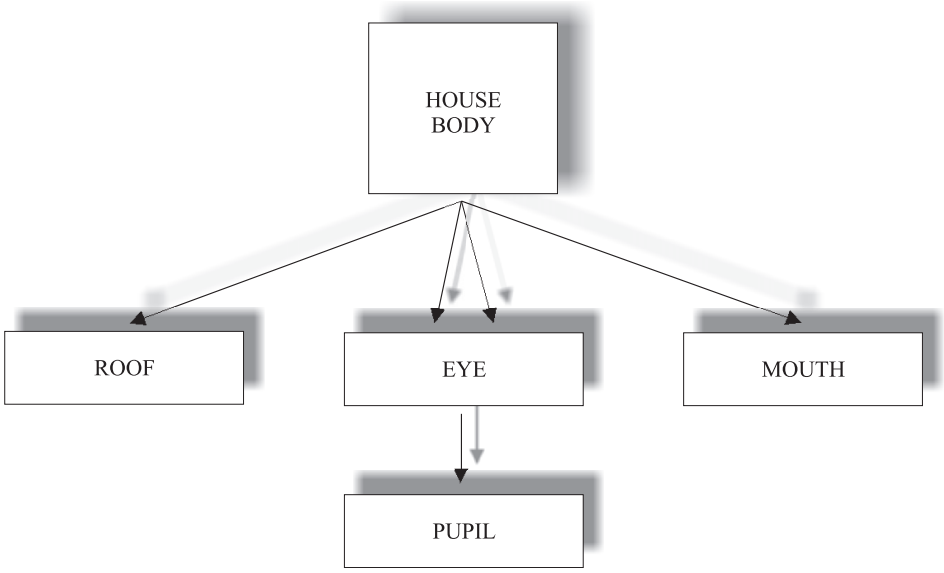

Figure 2.13: Movements hierarchy.

2.4.3 Animation

We can deform the objects by means of HENLT, but these deformations are statics, in the sense that for every configuration of HENLT an image is obtained. Animation is produced with a sequence of transformations. The use of HENLT provides an easy extension to achieve this.

As deformations are defined by control functions, they can be made to evolve over time (Figure 2.15). The parameters that define the control functions can be represented by means of *movement* functions which describe their variation.

For example, given a B-spline curve defined by four points, the position of every point can be described by four movement functions which may be of any type. In fact, it is possible the surface that represents the function's position may be not a B-spline surface. For example, the function controlling the first point could be linear, and the function controlling the second point a sine. This curve allow us to obtain the dynamic of the parameters which define the control functions. These movement functions depend upon a function that synchronizes everything, acting like a master clock.

The used functions are monoevaluated and two dimensional, that is, $y = f(x)$. The general idea of the evolution of the functions is that they are defined using a set of parameters which can be variable. This variation can be related to function, which itself can have it parameters variable and so on, until reaching an initial function that does not depends of other.

Definition 6 *A control function f is a triad ($g(\alpha)$, abscise-range, ordinate-range), being g the function that determines the behaviour of the deformation, depending upon α, and abscise-range and ordinate-range given by two values $Min_{abscise}$ and*

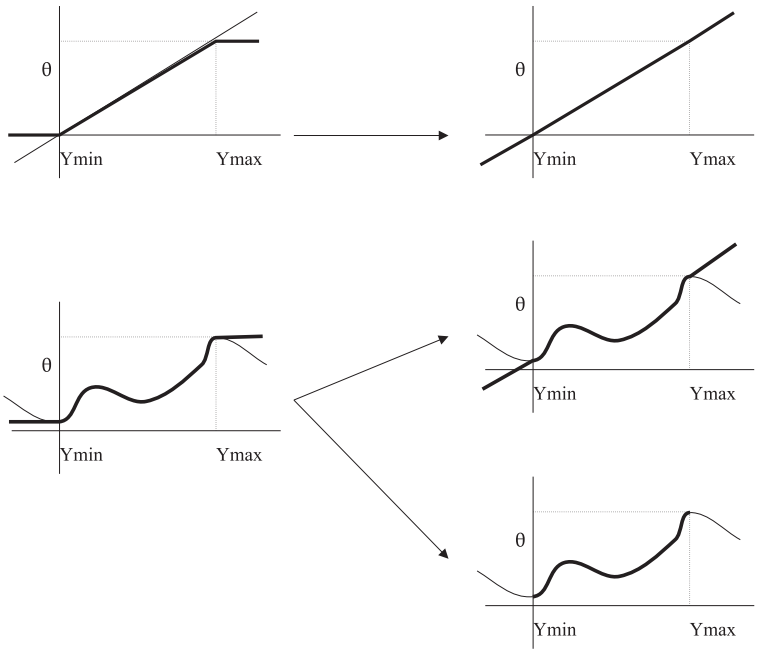

Figure 2.14: Posibilities of extension in control functions.

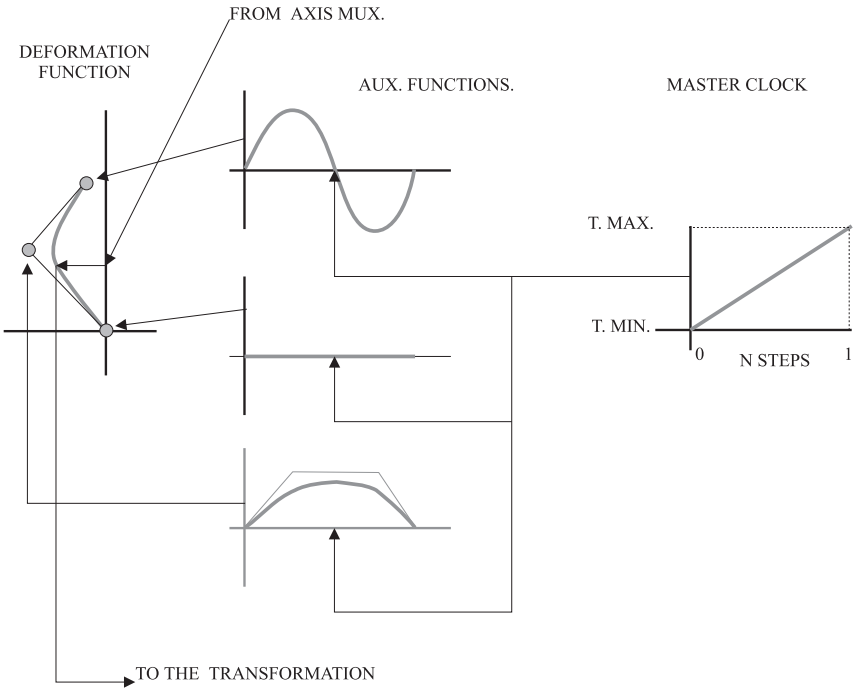

Figure 2.15: Functional-temporal dependence of a control function.

$Max_{abscise}$, and $Min_{ordinate}$ and $Max_{ordinate}$ respectively.

Ranges are determined by the definition of HENLT, because it is necessary to define a validity range for the independent variable, as well as, a application range. The possibility of referencing other functions appears in that the ordinate-range values Max and Min can depend on other functions. That is, $Min_{ordinate} = f_1(s)$ and $Max_{ordinate} = f_2(t)$, being f_1 y f_2 functions. The other possibility is that range values to be constants, $Min_{ordinate} = K$ and $Max_{ordinate} = L$, with K y L constant values. It is possible any combination of constant and dependent values. Maybe that the more important characteristic of control functions is that α can be dependent on other function. In this case, $\alpha = f(\beta)$, which itself is dependent of other function and so on. In such case, the abscises are not defined, because they are imposed by the ordinates of the function which it depends to. A stop condition must be imposed. This is achieved with a function which has the next definition $f_{basica} = (y = x; 0 - 1, Min_{ordinate} - Max_{ordinate})$. By default, $Min_{ordinate} = 0$ and $Max_{ordinate} = 1$ but they can be adjusted to the wanted values.

Chapter 3

Implementation with polygonal models

In this chapter there is description of the solutions that have been implemented and their relationship with the general exposition given in the previous chapter.

3.1 3D model

The selected 3D geometric model was a boundary representation one based in flat polygons, specifically triangles, using the Winged-Edged data structure [21, Pages 545-546].

In general, the polygonal b-rep models represent objects as an approximation. The geometry of the object is described by means of vertices than belong to the surface, and the topology is represented with edges and faces. Currently the representation of objects using triangles is very common. In fact, it is considered as a "low level" representation, because is used as a goal for other "higher level" representations, as for example parametric surfaces, implicit surfaces, and so on. This is due to is simplicity. There are many developed techniques for flat polygons, specially triangles, as for example Gouraud and Phong shading, active edges filling, z-buffer [21, Pages 668-672], and so on. There is also special hardware, even at PC levels, that accelerate the rendering process. Finally, there are libraries that use triangles as basic graphic primitives, for example OpenGL.

Another characteristic is that the normal vector can be easily evaluated, in contrast to other kind of representation (Page 18).

The problem with this model is that normally, is necessary many faces for obtaining a good approximation (Figure 3.1). This representation is also discrete which produces aliasing effects. For example, it is difficult to determine which is the silhouette in the Figure 3.2, and the result is only an approximation. The more serious problem occurs when the model suffers transformations that can change is topology, for example a big twist. This problem must be controlled by the user.

The main reason that takes us to select the triangle based polygonal model is that allows the detection and classification of geometric elements that will form the sketches in a easy way. In this case, edges will be sketches. So, we need to adapt the

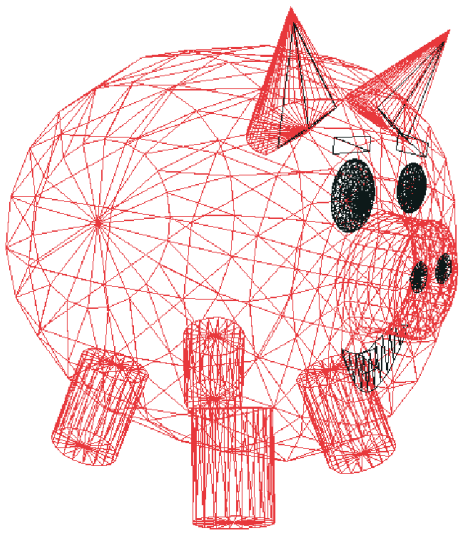

Figure 3.1: A polygonal model with a low number of faces.

definition of *Virtual Lights* to a non-continuos representation. The animation does not need to be adjusted, because the selected method is representational independent.

3.1.1 Objects creation

The data estructure used is the winged-edged one (Figure 3.3). The data structure used can be seen in Figure 3.4. The objects and inheritance structures are shown in Figure 3.5.

The reason of using the winged-edged data structure is that allows to easy some searching for operations. The implemented system only allows two-manifold objects.

The objects can be obtained in several ways:

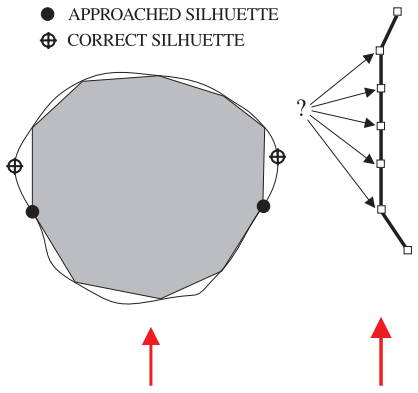

Figure 3.2: The problem of selecting edges that will form external sketches.

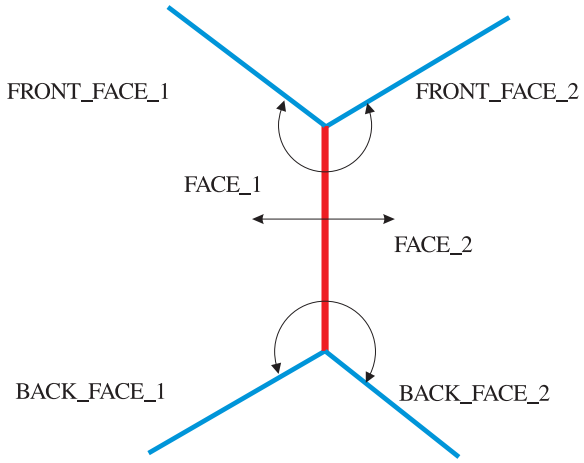

Figure 3.3: Winged-edged data estructure.

- To create mesh-like surfaces.
  - Introducing points by hand.
  - Obtaining the points randomly.
  - With a B-spline surface from which the points are sampled to form a mesh.
- To create objects by revolving through an axis.
  - Introducing points by hand.
  - With a B-spline surface from which the points are sampled to form the curve.

3.2 Sketches

The adjusted *Virtual Lights* model will be show after exposing the sketches generation in a more detailed way.

for all Objects **do**
  Classification and selection of edges
  Computing intersections between edges
  Forming chains of edges
end for
Visualizing edge chains

In the first step, the classification and selection of edges that will be part of sketches must be done. The process is:

for all Virtual lights **do**
  Computing the reflection of both faces
  if pass the conditions for the virtual light **then**
    To mark the edge as an sketch

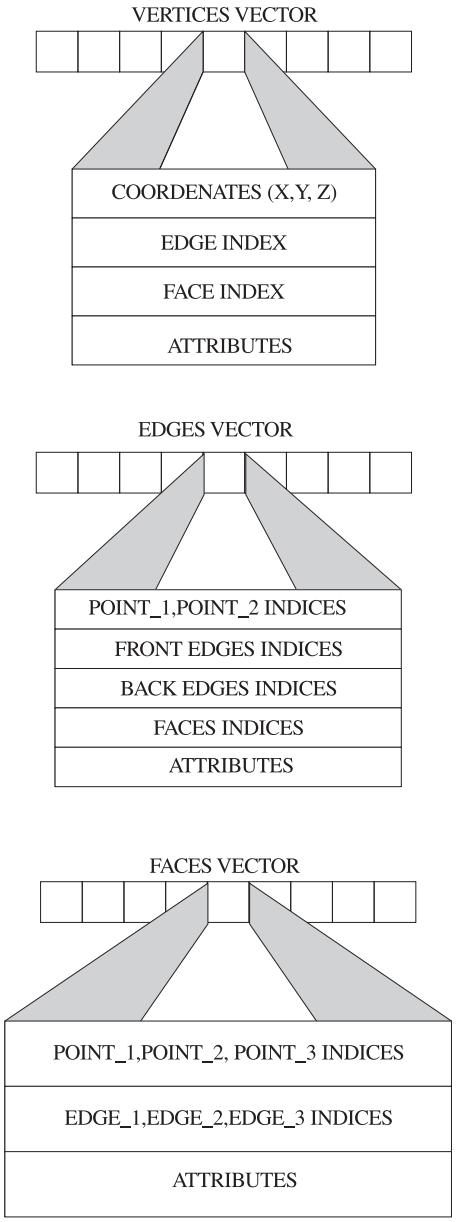

Figure 3.4: Data structures.

end if
end for
Global operation

In Figure 3.6 can be seen the stage of the algorithm, which are commented below.

The first step detects which edges will be part of the sketch. For every edge, the reflection values of every face in relation to every associated virtual light are computed. Faces can be classified as *visible* and *invisible*, depending on they are visible or not from the virtual light position (the formalization of this classification can be seen in page 47). Edges are classified *normal*, *visible_ sketch* and *invisible_ sketch*. One edge is normal if it is shared between two faces that are visible or invisible. Normal edges can not be part of a sketch. An edge is part of a sketch when it belongs to a

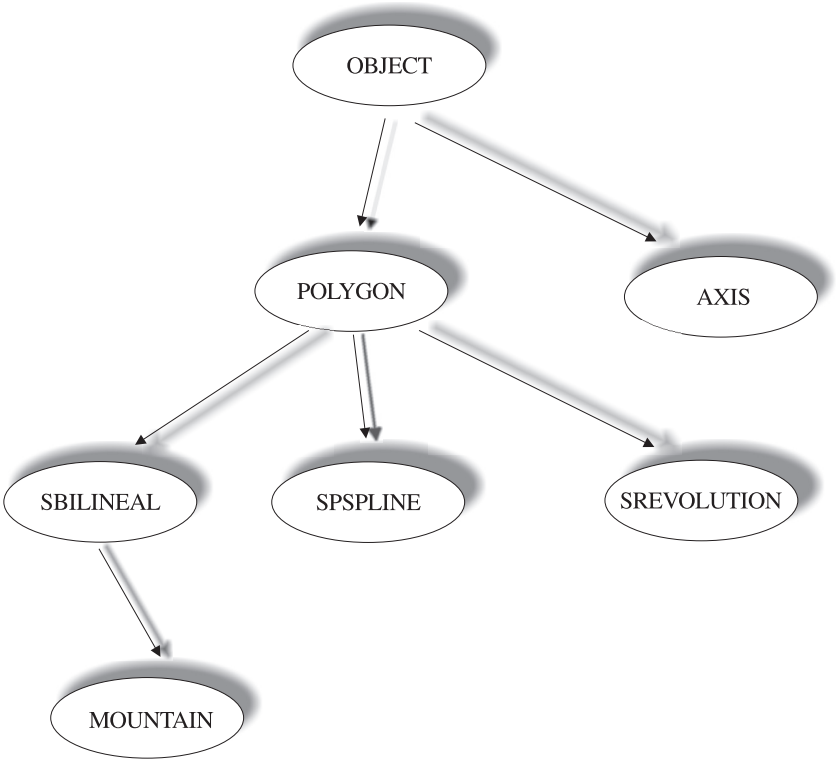

Figure 3.5: Inheritance structure.

visible face and an invisible face at the same time. Depending upon the relative positions of the faces, edges can be defined as visible_sketch (henceforth only sketch), which are edges whose visible face is in front of the invisible one; invisible_sketch edges are those whose invisible face is in front of the visible one. Faces are also classified depending upon they have or not one or some sketch or invisible_sketch edges.

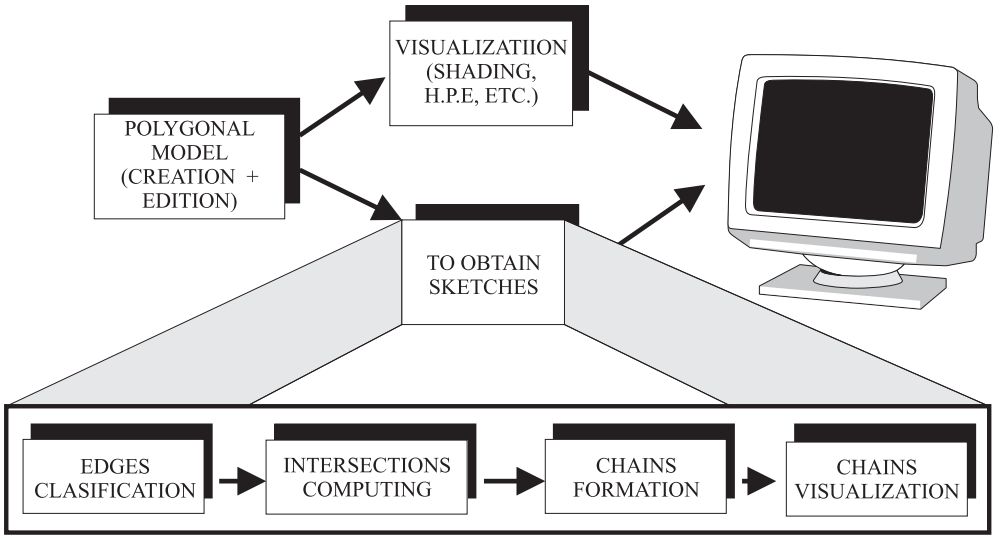

Figure 3.6: Main procedure structure.

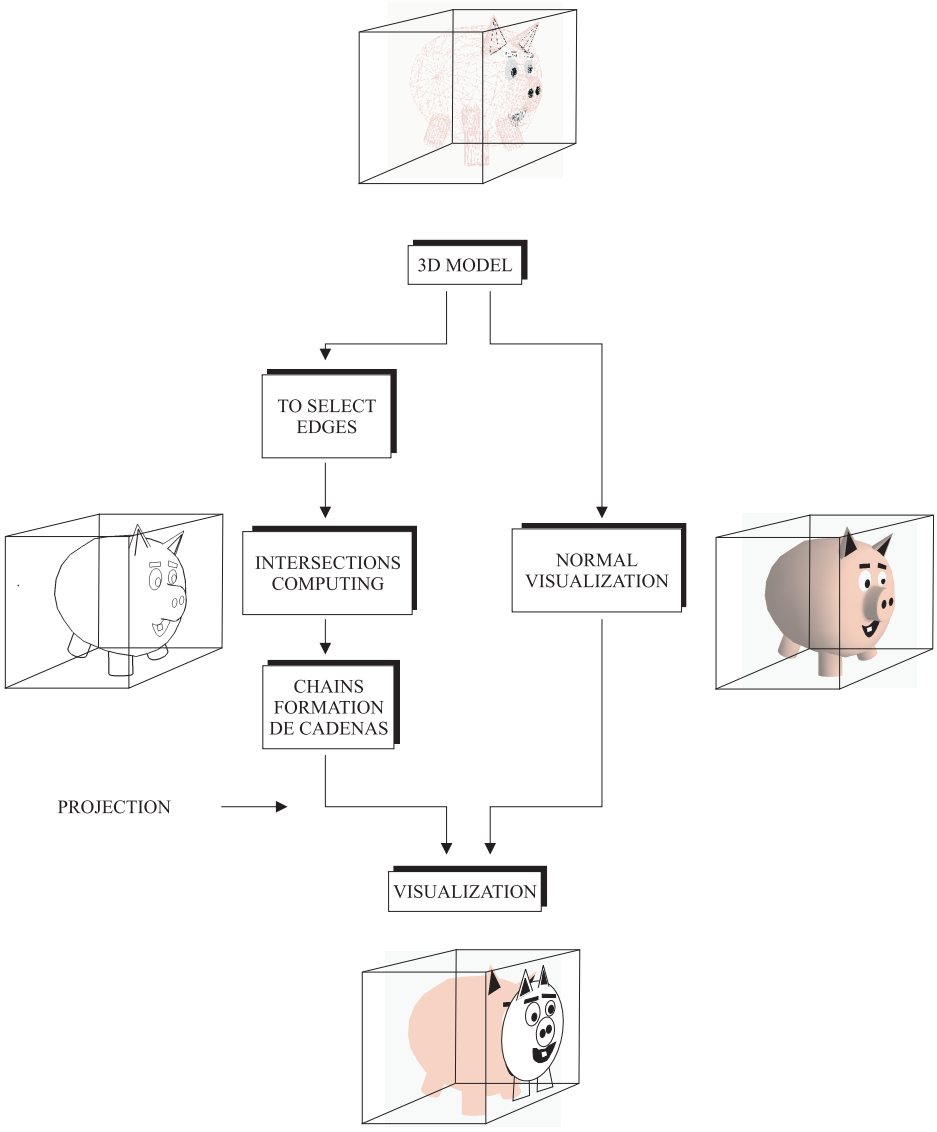

Figure 3.7: Visualization mode 1.

The face that has some sketch or invisible_sketch is called sketch_face. Otherwise, the face is called normal_face.

One of the virtual lights will allow us the obtention of silhouettes. In general, this virtual light is unique for all objects, though there is the possibility of defining other virtual lights for obtaining special effects. Once the reflections are computed the edges are classified depending on the defined conditions of the virtual light. Then, the partial results are combined with logical operations for producing the final result.

In the second step, the algorithm computes the intersections between the projections of the faces. The algorithm must obtain not only what is in front and what is in back but also how this happens, in such a way that it allows us a great flexibility in the visualization process (to change the thickness of lines, and so on).

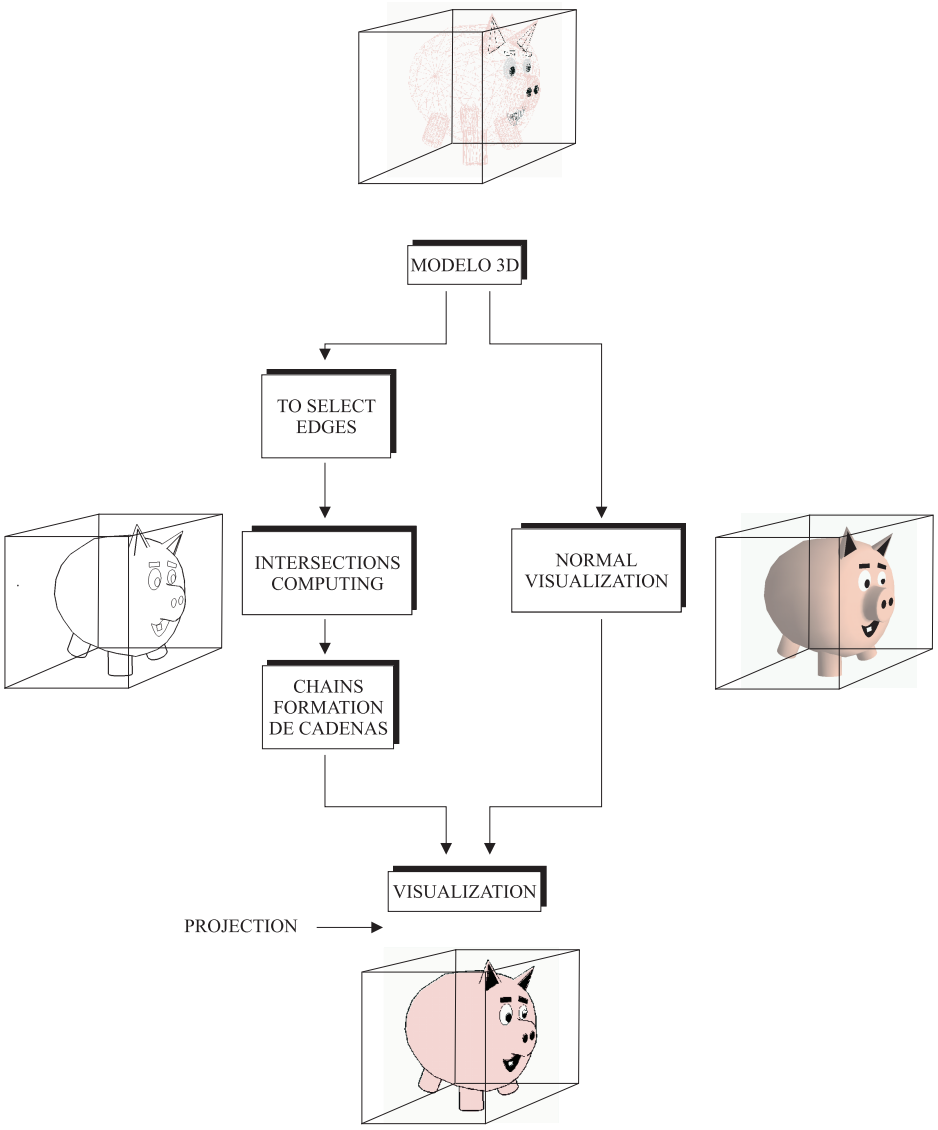

Figure 3.8: Visualization mode 2.

The solution is based in the next idea: to find intersections between edges is similar to find intersections of borders of faces. Though it is simple and obvious, the difference is the type of element that is used: in one case edges and in the second case faces. The great advantage in working with faces is that, unlike edges, the depth of a point can be classified even if there are not intersections. This situation happens when there are sketches that are inside others. Using faces is possible to determine if sketches are visible or not. It is possible to use ray tracing or other methods, but with complementary elements for not producing incorrect results. The computation is achieved in object coordinates. The problem of the method is that has quadratic complexity.

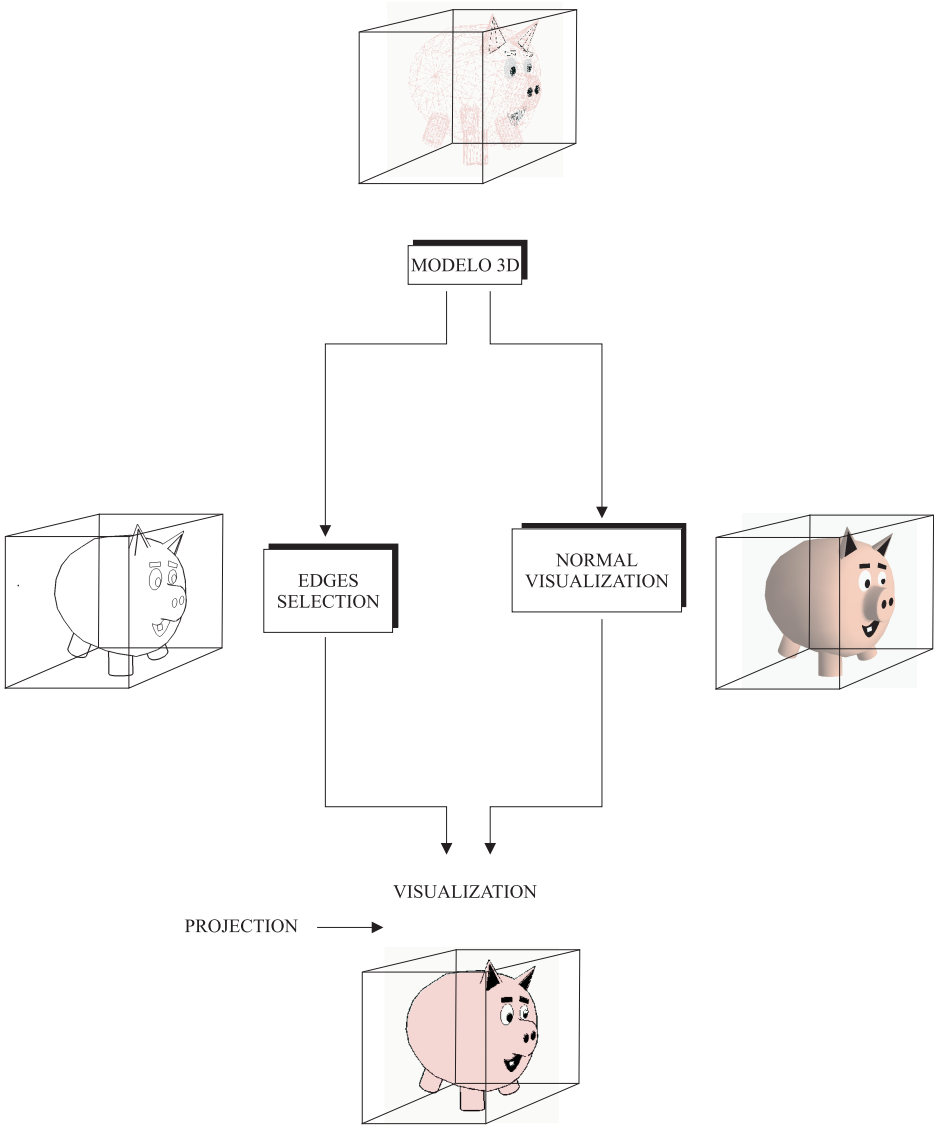

Figure 3.9: Visualization mode 3.

In the third step, edges can be drawn with the obtained information, but we are interested in form chains of closed and opened edges. This has the advantage that points can treated as control points of curves and textured surfaces, which is very interesting for illustration. This is the reason for using objects coordinates; we do not want chains to be broken at the border of the window. If there is more than one candidate for being part of the chain, there are two options which can be selected: it can be selected first the interior one or the exterior one. The great advantage of forming chains is that they can be treated as unique elements with their own attributes instead of individual edges, everyone with its attributes.

In the last step, chains are drawn using the attributes. There have been implemented three modes of visualizing chains, which defer in complexity and goals.

The most complete mode (Figure 3.7) extends the Appels's quantitative invisibility concept in the sense that it also controls how many times an edge is in front of one or several faces. This concept can be called *quantitative visibility*. It adds one more grade of flexibility in visualizing sketches.

The chains can be visualized in different ways (Figure 3.10). This method can also be used as a hidden line elimination method. When this method is used, OpenGL visualizes all objects except sketches, using the z-buffer. Then, sketches are drawn in front of all objects (with z values equal to the front plane value).

The second mode is a variation of the previous one. The formed chains are transformed in the same way as objects (Figure 3.8). The z-buffer is applied to all objects and sketches. The visual characteristics are maintained, using the pipeline of OpenGL. This allow that every chain has its own color.

In the basic mode, the edges are treated as single elements (Figure 3.9). It is very fast but it has not the same degree of flexibility that the previous modes.

3.2.1 Virtual lights

The model that was exposed in the previous chapter was a general description of the virtual lights mechanism, without being associated to any concrete geometric model. Here, the model is explained under the focus of its application to a 2-manifold triangle based model.

The general definitions must be adjusted to the polygonal model, passing from a

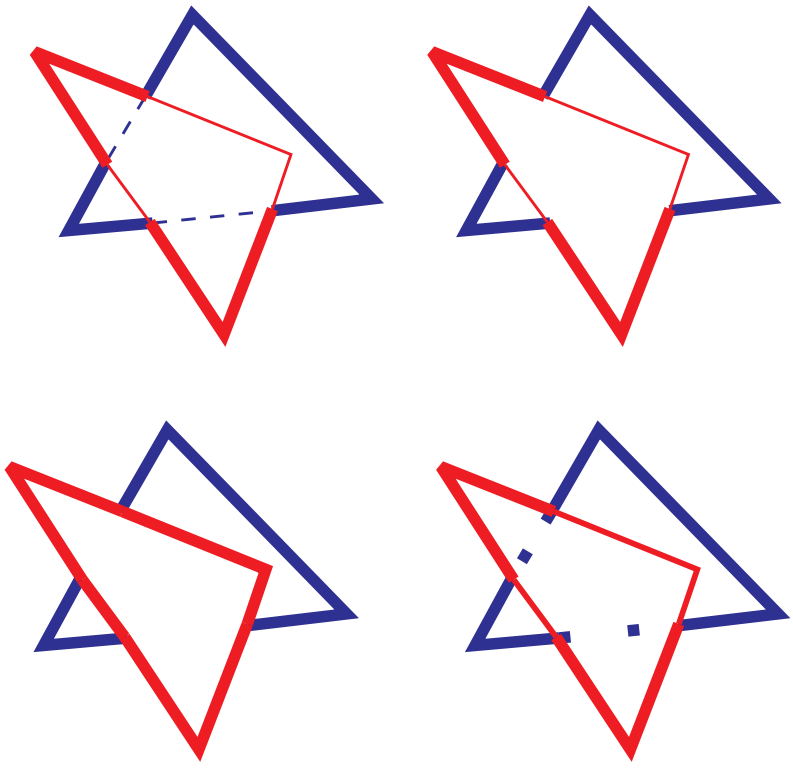

Figure 3.10: Forms of visualizing sketches.

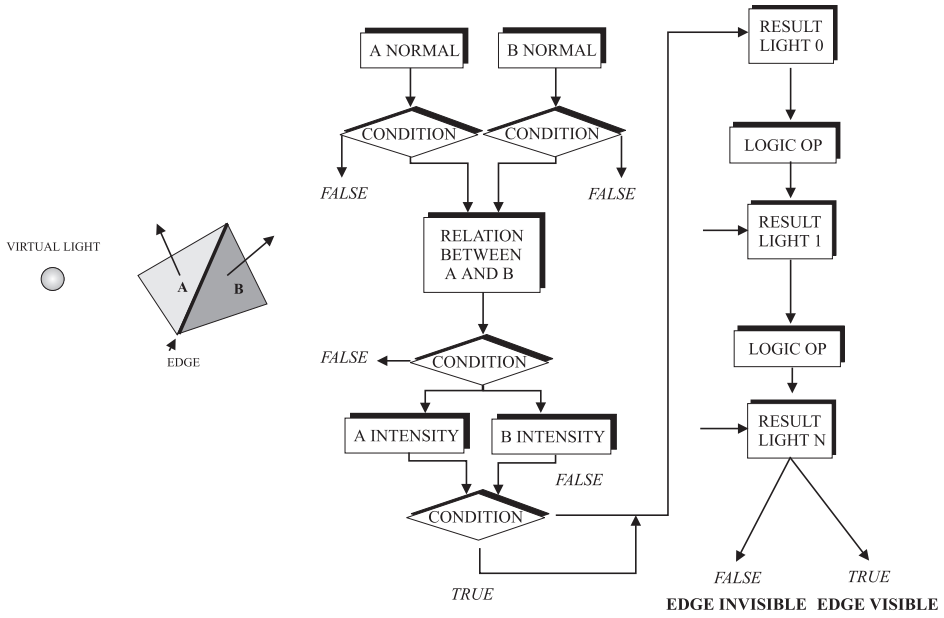

Figure 3.11: Reflection computing procedure using virtual lights.

continuos form to a discrete one.

The definition of external and internal sketches is given below, but previously we need to define the meaning of visibility and invisibility of a face:

Definition 7 *A face is visible if the dot product between the normal vector of the face and the direction of the* virtual light *is greater than 0. Otherwise, the face is invisible.*

This is the normal definition of visibility. Where the *virtual light* is at infinity, the direction is given directly by its definition. Where the *virtual light* is local, the direction must be computed from the difference between its position and any point on the face.

Definition 8 *Given a 2-manyfold 3D polygonal model and a* virtual light*, an external sketch is an edge which has one face illuminated (visible) and the other face non illuminated (invisible).*

When we want to obtain the same silhouette than the one seen by the observer, the virtual light must be located the same position as the observer. This is the procedure that allows us to treat both types of sketches in the same way.

Definition 9 *Given a 2-manyfold 3D polygonal model, a* virtual light*, and three intervals of real numbers* $IC(ic_1, ic_2)$*,* $DC(dc_1, dc_2)$*,* $RC(rc_1, rc_2)$*, an internal sketch is an edge that satisfies the following conditions:*

(a) The angle that forms the normal vector of both sides and the virtual light *is at the interval* IC*.*

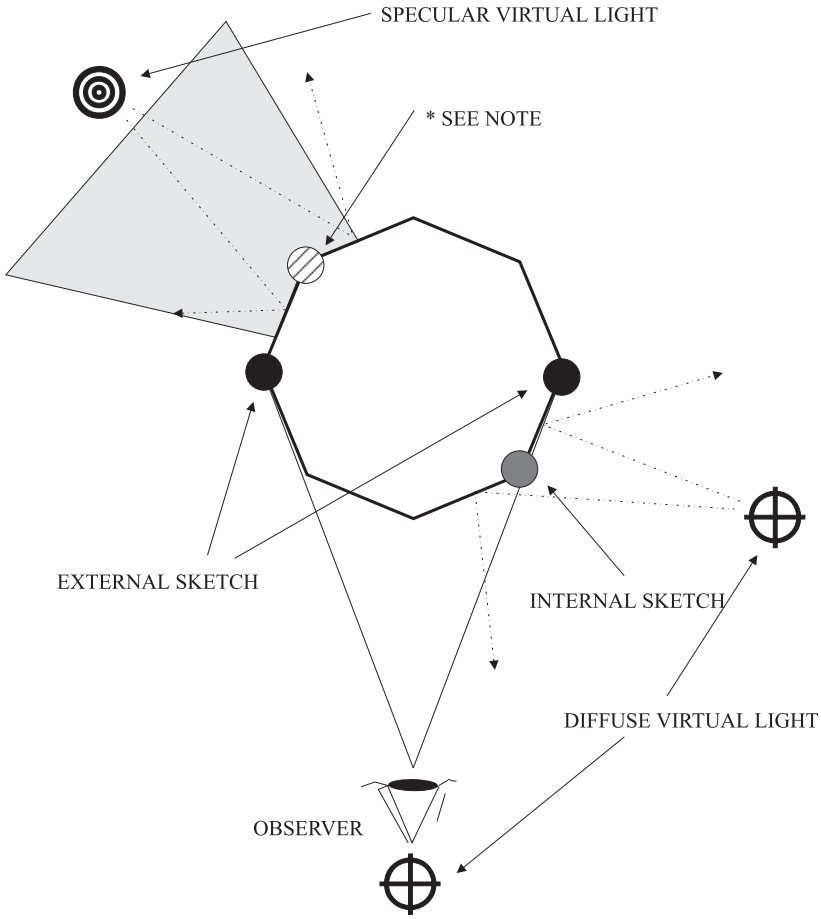

Figure 3.12: Scheme of application of virtual lights. * This edge will be a sketch when the observer comes into the gray zone.

(b) The angle between the normal vector of its two faces is at interval DC.

(c) The computed difference of reflections, which are obtained using the illumination model is included at interval RC.

Conditions (a) and (b) can be applied to obtain the external sketches. Condition (b) allows the situation where the faces have a more concave or convex configuration to be selected. Objects have a set of intervals for every defined *virtual light*. So, do we have not only different types of lights but also different conditions for each one, which makes the method very flexible for obtaining sketches where and when the user wants them. The process is shown in Figure 3.11. In Figure 3.12 there is an example with virtual lights and who their positions affect to edges.

Edges have an attribute for every defined virtual light. This attribute can have values that indicate if the edge is alway a sketch, it is never or depends upon the operations with virtual lights. This mechanism allow that, for example, when two objects are joined, their respective external sketches to be eliminated.

The detailed data format of virtual lights is exposed in Appendix A, section A.2.

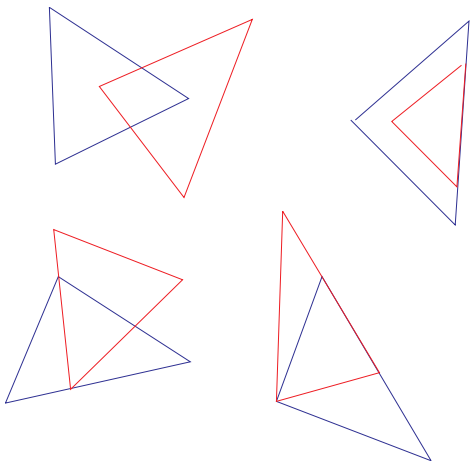

Figure 3.13: Coincidence of vertices.

3.2.2 Obtention of intersections

Once the edges that will form sketches are detected, the intersection between them must be computed, as exposed in the general algorithm (Page 41). In visualization mode 1 (Page **??**), not only the intersection must be computed but also is necessary to detect sketches that have not intersections and that can be visible or invisible. The computing of intersection between edges is achieved computing the intersections between sketch type faces with the others, that is, the rest of sketch type and all normal type faces. This makes that the process has quadratic complexity. The computing can be decomposed in a simple process of evaluating intersections between two faces. Applying this method to all the affected faces, the final goal is reached.

We expose the problem of computing intersections between two triangles below. Some conditions must be imposed:

1. Faces have to be convex
2. Faces have to be plane
3. Interpenetration is not allowed
4. All faces are defined in a direction
5. Two faces can not be co-planar and intersect in their projections. Several degenerated cases are resolved with agreements
6. Two edges of the same polygon can not be co-linear. This is guaranteed with non degenerated triangles.

Formally, the geometric elements used by the algorithm are polygons of triangular shape, because they can be converted in polygons with more than three vertices.

The algorithm must be able to operate with any configuration of two triangles, normal cases and degenerated cases (Figure 3.13), several triangles sharing a sketch type edge (Figure 3.14), and cases in which there is coincidence in the projection of

several edges, of any type. This last case can happens with 3D opened and closed objects (Figure 3.15). Although faces that share an edge, also share the extreme vertices, it is not the same with the attributes of those vertices, depending on they are take as belonging to one or the other face

It is necessary to define some concepts to understand how the algorithm works. The visibility change concept is similar to Appel's quantitative invisibility, but with something that is different. The implemented visibility change counts how many times a vertex is in front of a polygon and how many is back. This information gives more flexibility in visualization.

Vertices and their visibility change attributes are the elements that define the edges. The geometry of an edge is define by its two extreme vertices, but the attributes are those of the first corrsed vertex, which can be one or other depending upon the direction in which the face is traversed. Formally, given two vertices, V1 y V2, the edged is defined as the half-open interval [V1,V2). So, a shared edge can be drawn in two way, because there is an attribute for each face.

Another important concept is constituent vertices and added vertices. Constituent vertices are those which form the initial triangle; the number does not change. Added vertices are those that are added to the original geometry when there are normal intersections (this concept will be seen below). The added vertices are only used in the update data phase.

The concepts of absolute reference and relative reference are also important. It is related with the visibility change concept in this manner: constituent vertices have absolute visibility change attribute while added vertices have relative visibility change attribute. Absolute values can be used directly. Relative values must be referenced to absolute values to be correct.

it is necessary to establish two agreement for resolving ambiguities. For example, between the triangles of case A in Figure 3.16 there are not intersections between edges, no one is inside the other, but there is a coincidence in one vertex projection, the vertex that is behind, and it is necessary to determine if it is inside or outside the other triangle. The agreement considers that is outside. Otherwise, the full edge

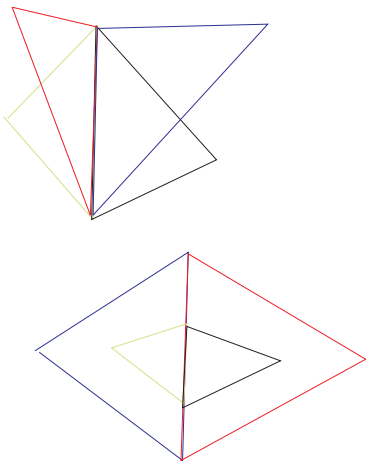

Figure 3.14: Sharing of an edge.

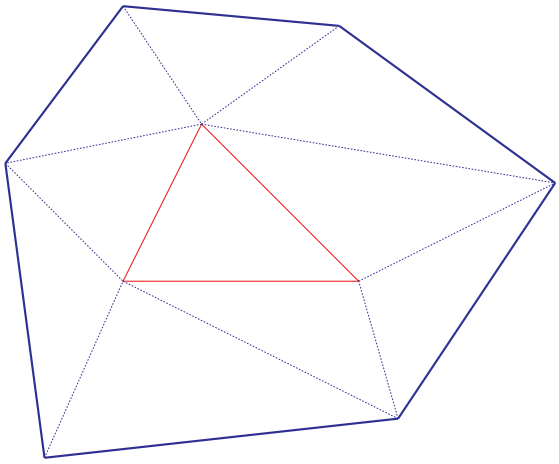

Figure 3.15: Coincidence in projection of several edges.

will be considered as interior to the triangle. This agreement also resolves the case when the coincidence between vertices is complete (the two vertices occupy the same position in space, case B, Figure 3.16).

Agreement 1 *If the only intersection between two faces is in one vertex, that vertex is outside, that is, there are not intersections.*

Another case is the coincidence between an edge and two or more triangles. This can happens because: a) the two vertices of edges have the same coordinates, b) one vertex coincides totally and the other coincides in projection, or c) the two vertices coincide in projection. The problem is in deciding when the edge of one face covers the edge of other face. If there is a spacial ordination the problem is almost resolved: one face is ahead and the other is behind (cases C and D in Figure 3.16). The E case is different because there are intersections. It can be solved checking the position of faces and establishing a spatial order.

In case F, Figure 3.16, there is total coincidence, there are not intersections, and there is no spatial ordering. If the objects are only 2-manifold is possible to establish

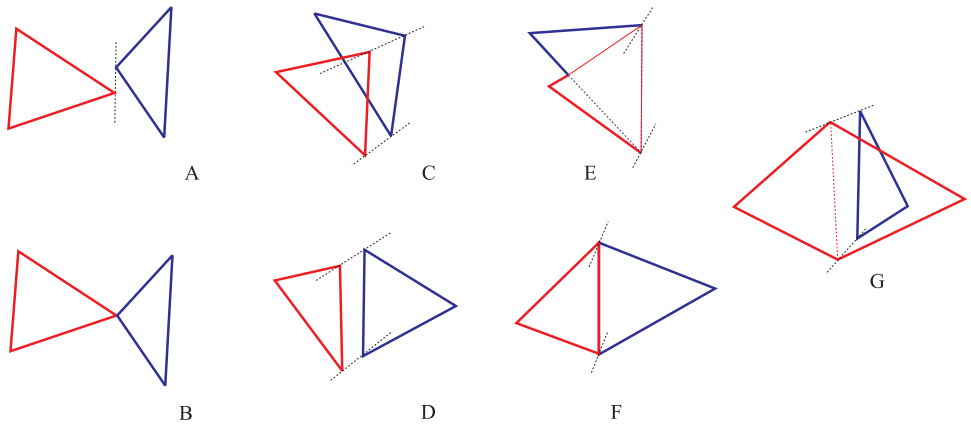

Figure 3.16: Several configurations of triangles.

a spatial ordering. But other problem appears, as can be seen in G case, where there is edge coincidence but it is not possible to establish a spatial ordering. The problem is that when we consider the A triangle in relation to B triangle, it indicates that A is behind B, and when the same operation is carried out between A and C, it indicate that A is behind C too; the final effect is that A is behind two objects and this in not true. It is resolved with an agreement:

Agreement 2 *When there is edge coincidence between two triangles, there will have interaction if both triangles are in the same side of the shared edge, else there is not interaction and an order is established.*

This agreement makes the algorithm more simple in treating some cases, the results have a high level of visual coherence and allow to work with non 2-manifold objects[1].

The information that the algorithm can obtain allow us that lines can have "character". This "character" in carried out making that sketches can have several form at the start, middle, and final part, and also changing the thickness and style, and so on (Figure 1.28). This variation depends upon the relative position of sketches. For example, the part of a sketch that is behind can be drawn with one style, and if the sketch passes ahead, to change the thickness, and so on. This flexibility can not be achieved with a hidden part elimination method.

Solution

The steps of the algorithm are:

1. To classify all possible cases that can appear between two triangles.
2. To compute intersections.
3. To update the values of geometry and attributes.

The fist step is a classification. Given the A triangle and B triangle the cases are:

$$\begin{array}{lll} A \cap B = 0 & & \\ A \cap B \neq 0 & \Rightarrow & A \subset B \vee B \subset A \\ & & A \not\subset B \wedge B \not\subset A \end{array}$$

We need to define what are normal and degenerated intersections before to explain the intersections computing.

A normal intersection is produced in the interior of two edges (Figure 3.17). Formally, given two edges, E1 with extreme vertices P1 and P2, and E2 with extreme vertices Q1 and Q2, a normal intersection is produced if : there is intersection, edges are not co-linear, and the intersection I complies that $P1 < I < P2$ y $Q1 < I < Q2$. When the three conditions are carried out the intersection is the same for both edges, at least in projected coordinates.

[1]The intersection method can do it, but for simplicity it is not allowed

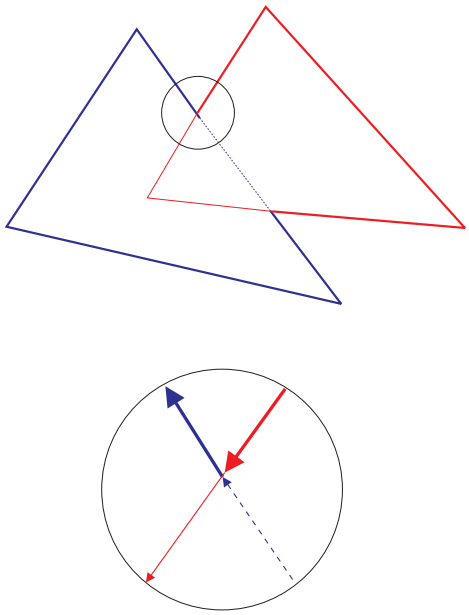

Figure 3.17: Normal intersection.

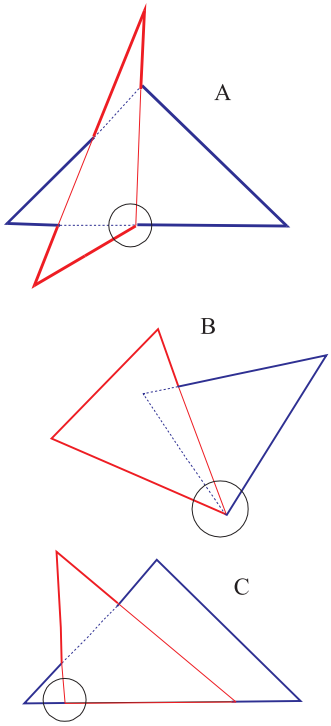

Figure 3.18: Degenerated intersection.

The visibility changes of edges are produced in intersections. These changes are: an edge come into (IN) a polygon or an edge come out (OUT) a polygon. There is a symmetry because when an edge come in the other come out. There is an attribute that not only marks the IN's and/or OUT's, but also if they are ahead or behind the polygon (Figure 3.17).

A special or degenerated intersection [39, Page 154] is produced in one or two vertices. In the first case there is an intersection between a vertex and the interior of an edge, and the second case occurs when there is an intersection between two vertices (Figure 3.18). The last one is the most difficult to resolve. Although there are intersections, they do not produce new vertices, but we have to control the visibility change, that is, the constituent vertices must be always absolute references. In this kind of intersection maybe symmetry, as commented previously (Figure 3.18, cases A and B), but is also possible that it does not exist, for example, for one face the intersection is an IN case, and for the other the vertex is inside the face (Figure 3.18, case C). If there is an IN, it makes that the corresponding attribute, ahead or behind, to increase in one. If the intersection is an OUT it is not necessary to change the attribute, because it represents a return to the previous state to the IN.

3.2.3 Detailed description of intersections computing

The algorithm uses triangles defined by a list of three vertices and a list of three edges. Every edge is composed of two vertices. This estructure allows us to introduce new intersections. The vertices, new and added, have, at least, two attributes: its relative position, and depth position in relation to the polygon. The first attribute can take the next values:

INSIDE
OUTSIDE
INSIDE_ALIGNMENT
OUTSIDE_ALIGNMENT
VERTEX_ALIGNMENT
IN
OUT

If a vertex is aligned, this is a temporary state, finally it has to be one of these values:INSIDE, OUTSIDE, IN or OUT. The second attribute can have the values AHEAD and BEHIND.

Classification of point positions from one triangle to the other

The first thing is to determine the relative position of every vertex of the triangle that is treated (called T1) in relation to the second triangle (called T2). The solutions are:

- All the positions are inside.
- Some or all the positions are outside.

The fist case indicates that the point is inside the triangle T2. The second one indicates that the vertex is outside of T2

There is also a subclassification that is necessary for the algorithm to work. This subclassification distinguish between a vertex that coincides with other, or is included inside the edge, or is aligned with the edge in the non-valid part (Figure 3.19).

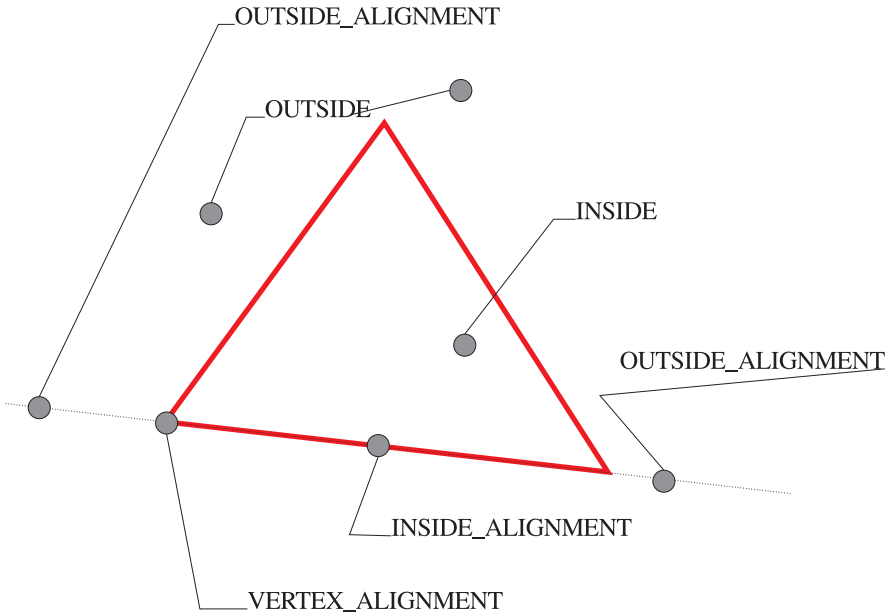

Figure 3.19: Positions of a vertex in relation to a polygon.

General position	Concrete position
INSIDE	INSIDE
OUTSIDE	OUTSIDE
OUTSIDE	OUTSIDE_ALIGNMENT
INSIDE	VERTEX_ALIGNMENT
INSIDE	INSIDE_ALIGNMENT

The above classification is applied to every vertex of the triangle. The possible results are:

- All vertices are inside → INSIDE
- All vertices are outside → OUTSIDE
- Some vertices are inside → INDETERMINATE

This gives us 9 possibilities:

Triangle T1	Triangle T2	Case
INSIDE	INSIDE	1
INSIDE	OUTSIDE	2
INSIDE	INDETERMINATE	3
OUTSIDE	INSIDE	4
OUTSIDE	OUTSIDE	5
OUTSIDE	INDETERMINATE	6
INDETERMINATE	INSIDE	7
INDETERMINATE	OUTSIDE	8
INDETERMINATE	INDETERMINATE	9

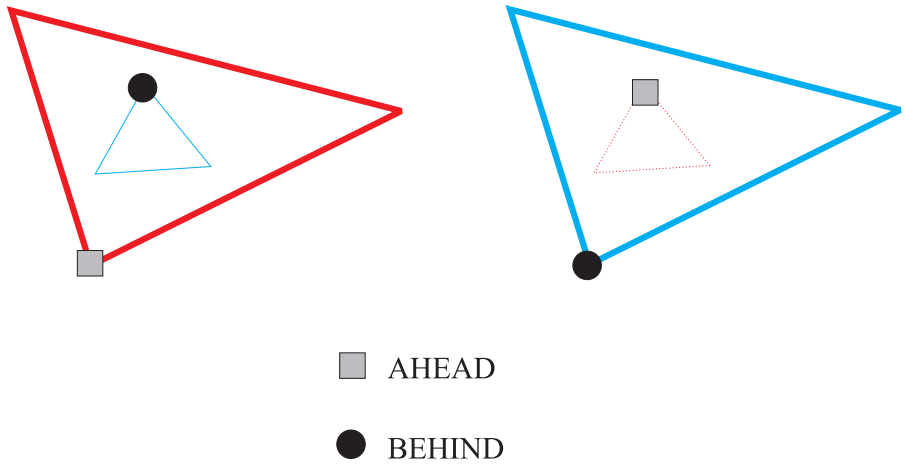

Figure 3.20: INSIDE-OUTSIDE and OUTSIDE-INSIDE conditions.

The case 1 indicates that there is a coincidence between the three vertices of the two triangles (al least, one intersection must be in projection, otherwise the triangles are co-plane). In this case, the relative position in depth of the two triangles must be determined. Case 2 and 4 are solved in the same way as the previous one (Figure 3.20). It can be seen that for cases 2 and 4 (also 1), there are no intersections. In these cases every vertex of one triangle are marked as INSIDE and vertices of the other triangle as OUTSIDE, and for the second case the other way round. The visibility is indicated by the values AHEAD and BEHIND.

The case OUTSIDE-OUTSIDE does not implies that there is not interaction between triangles, as can be seen in Figure 3.21, so it must be determined if there are intersections. Otherwise triangles are separate ones.

Cases 3 and 7 are not possible. In the rest of cases there are new intersections that have to be computed.

Detection and computing of intersections

The algorithm works in a similar way to the Sutherland-Hodgman's polygon clipping.

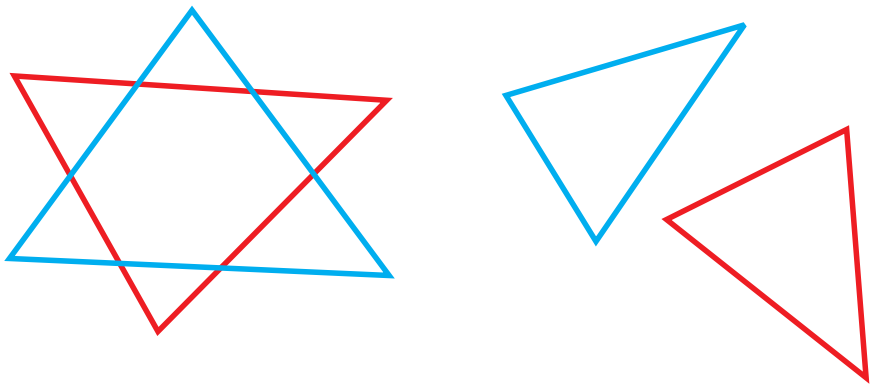

Figure 3.21: OUTSIDE-OUTSIDE cases.

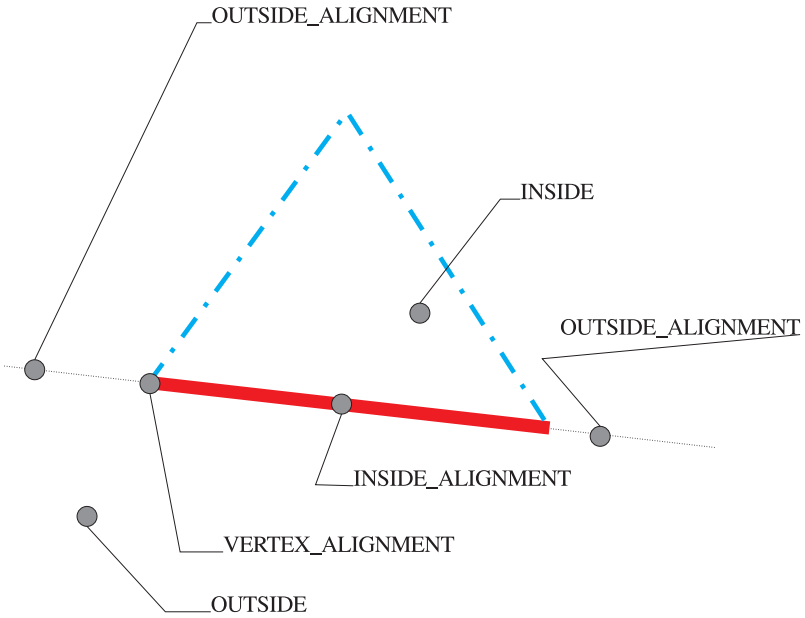

Figure 3.22: Positions of one vertex in relation to an edge.

For every edge of triangle T1, the constituent vertices of T2 are classified. The edge works like a window side. There is a classification, inside and outside, but also the subclassification indicating if there is an alignment.

The possible cases for a vertex and an edge are (Figure 3.22):

- INSIDE
  The vertex is in the inside side.

- OUTSIDE
  The vertex is in the outside side.

- VERTEX_ALIGNMENT
  The vertex coincides with other vertex of the second triangle. This can only be the first vertex of the edge. If it is the second vertex, it is not treated now.

- INSIDE_ALIGNMENT
  The vertex is aligned in the interior of the edge of the second triangle.

- OUTSIDE_ALIGNMENT
  The vertex is aligned in the exterior of the edge of the second triangle.

Once the points are classified, they are traversed sequentially, and depending on the current and next positions it can be determined if there is an intersection (this case is called simple or edge intersection). Sometimes is necessary to know the previous position (this case is called complex or vertex intersection).

In Figure 3.23 is described the algorithm of obtention of intersections.

```
for all Triangle T1 sides do
  To obtain the relative position of every vertex of triangle T2
  for all Vertex of triangle T2 do
    Depending on vertex position do
      INSIDE:
      Depending on next vertex position do
        OUTSIDE:
        if there is intersection then
          To compute the intersection
        end if
      endDepending
      OUTSIDE:
      Depending on next vertex position do
        INSIDE:
        if there is intersection then
          To compute the intersection
        end if
      endDepending
      ALIGNED:
      if vertex is interior aligned then
        Depending on next vertex position do
          ALIGNED: To compute the intersection
          INSIDE:
          Depending on previous vertex position do
            ALIGNED | OUTSIDE: To compute the intersection
          endDepending
          OUTSIDE:
          Depending on previous vertex position do
            ALIGNED |INSIDE: To compute the intersection
          endDepending
        endDepending
      end if
    endDepending
  end for
end for
```

Figure 3.23: Algorithm for obtaining intersections.

Intersections in edges

In this case, one vertex is INSIDE and the other is OUTSIDE or OUTSIDE and INSIDE. It has to be checked that the intersection occurs in the valid range.

Formally, given the edge A1 defined as [V1,V2) and edge A2 defined as [V3, V4), the valid intersections are those that occur between (V1,V2) and (V3, V4) (Figure 3.24). it must be checked that there are not aligned vertices.

There are visibility changes in all intersections, but we are interested not only in knowing the visibility, but how this visibility change occurs, if it is when the edge come into the face or when it come out the face. The first case is called IN and the

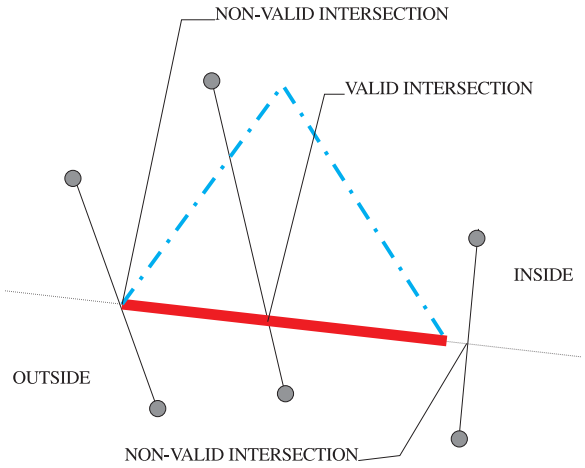

Figure 3.24: Valid and non valid intersections inside an edge.

second is called OUT (Figure 3.25).

It is also necessary to indicate the relative position. The attribute can have two values: AHEAD and BEHIND. So, for every intersection there are four possibilities: IN/AHEAD, IN/BEHIND, OUT/AHEAD and OUT/BEHIND, which are shown in Figure 3.25. With this information is possible to implement the visibility changes in visualizing sketches.

In Figure 3.26 are shown the case that can occur, pointing to if there is or not a new intersection.

Intersections in vertices

The reason for dividing intersections in two class is due to in vertices intersection it is not necessary to duplicate vertices.

There are three possibilities of alignment. The only one that produces a new in-

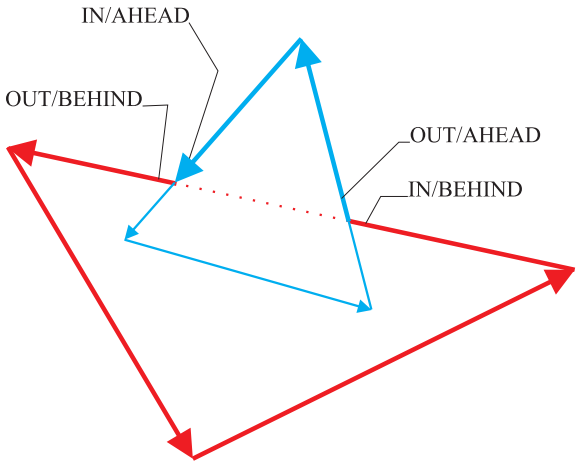

Figure 3.25: Intersections attributes.

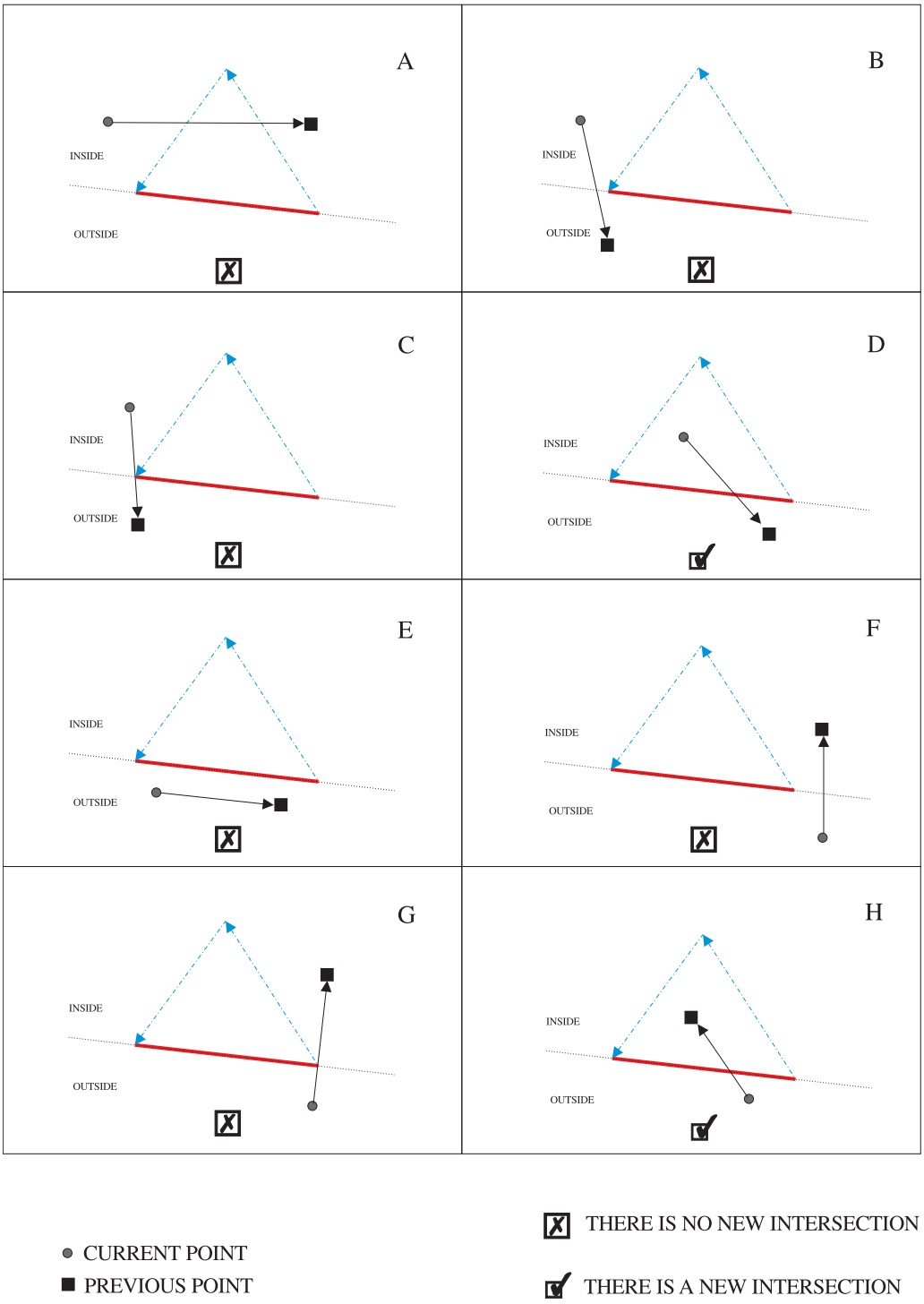

Figure 3.26: Possible cases in edge intersections.

tersection is INSIDE_ALIGNMENT. If the vertex is classified as VERTEX_ALIGNMENT there is not new intersection but is necessary to indicate if there is a visibility change. If the vertex has the value OUTSIDE_ALIGNMENT, there is no change. The possible case are shown in Figure 3.27, pointing if there is or not an intersection.

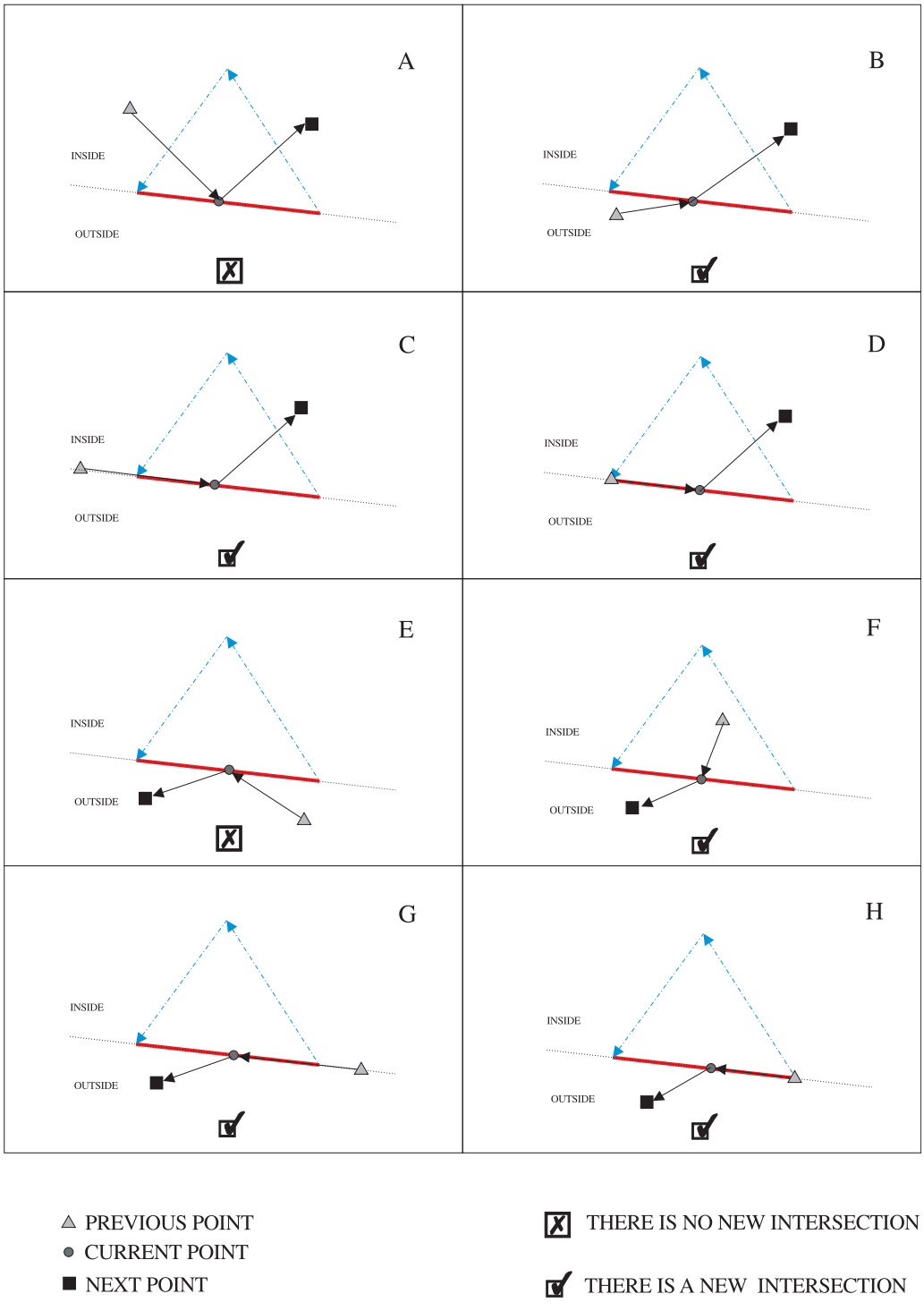

Figure 3.27: Cases in which there are new intersections.

The new intersections occur only in certain conditions, as can be seen in Figure 3.27. That is, when the next vertex is also aligned. When the vertex is INSIDE the previous vertex state must to be determined: if it is OUTSIDE or ALIGNED, then there is a new intersection. There is a new intersection if the previous vertex is INSIDE or ALIGNED, and the next vertex is OUTSIDE.

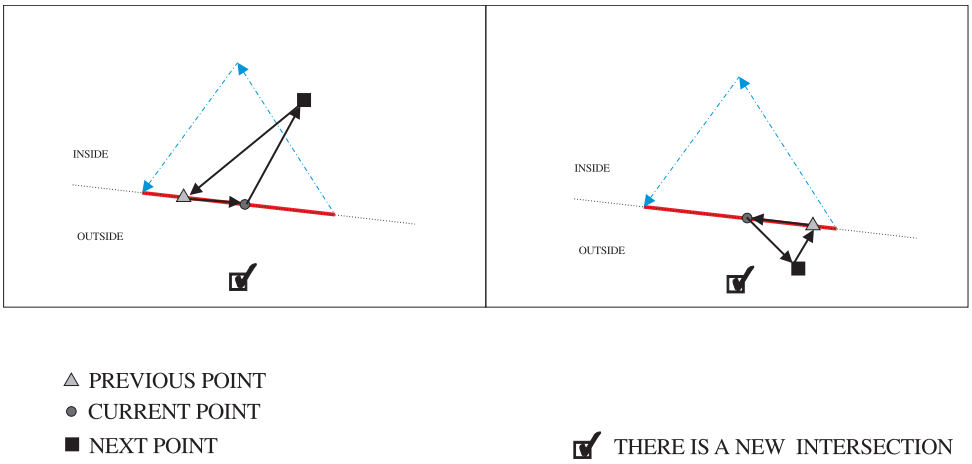

Figure 3.28: Alignment of two edges.

The OUTSIDE_ALIGNMENT is necessary for the algorithm to work, as can be seen in cases E y G in Figure 3.27. When there are two alignments (Figure 3.28), the case is correctly treated, computing two intersection for one edge and none for the other. It can be seen that if vertices are classified as OUTSIDE instead of ALIGNED_OUTSIDE the result is wrong.

The vertices position is also used to determine the visibility changes in new intersections. This is shown in the next table (counter clock wise):

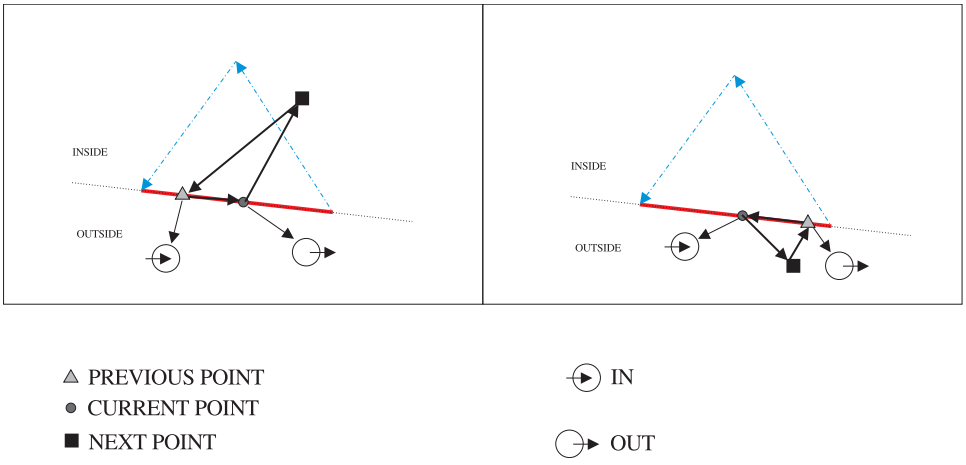

Figure 3.29: State change in aligned edges.

Previous vertex	Current vertex	Next vertex	State
INSIDE	INSIDE_ALIG	INSIDE	
INSIDE	INSIDE_ALIG	OUTSIDE	IN
INSIDE	INSIDE_ALIG	INSIDE_ALIG or OUTSIDE_ALIG or VERTEX_ALIG	IN
OUTSIDE	INSIDE_ALIG	INSIDE	OUT
OUTSIDE	INSIDE_ALIG	OUTSIDE	
OUTSIDE	INSIDE_ALIG	INSIDE_ALIG or OUTSIDE_ALIG or VERTEX_ALIG	OUT
INSIDE_ALIG or OUTSIDE_ALIG or VERTEX_ALIG	INSIDE_ALIG	INSIDE	OUT
INSIDE_ALIG or OUTSIDE_ALIG or VERTEX_ALIG	INSIDE_ALIG	OUTSIDE	IN

In Figures 3.29 and 3.30 can be seen the visibility changes represented in the table.

Update of data

Once is determined that there are new intersections or interaction between the triangles, the relative depth positions of both triangles are determined. The result is that one is AHEAD and the other is BEHIND. The update phase uses this information. The achieved actions are:

1. Determining the visibility change that VERTEX_ALIGNMENT vertices can have.

2. Introducing the new intersections in the data structure, including depth information.

3. If necessary, updating the values of all constituent vertices, including the aligned ones.

Given a vertex, the needed information is about there are or not intersections in the two edges that share the vertex. In case that there are two intersections the visibility does not change, so we are interested when there is only one intersection. For doing this classification, INSIDE_ALIGNMENT and VERTEX_ALIGNMENT states are considered as INSIDE, and the OUTSIDE_ALIGNMENT state is considered as OUTSIDE.

The table that controls this process is (V=vertex, I=intersection):

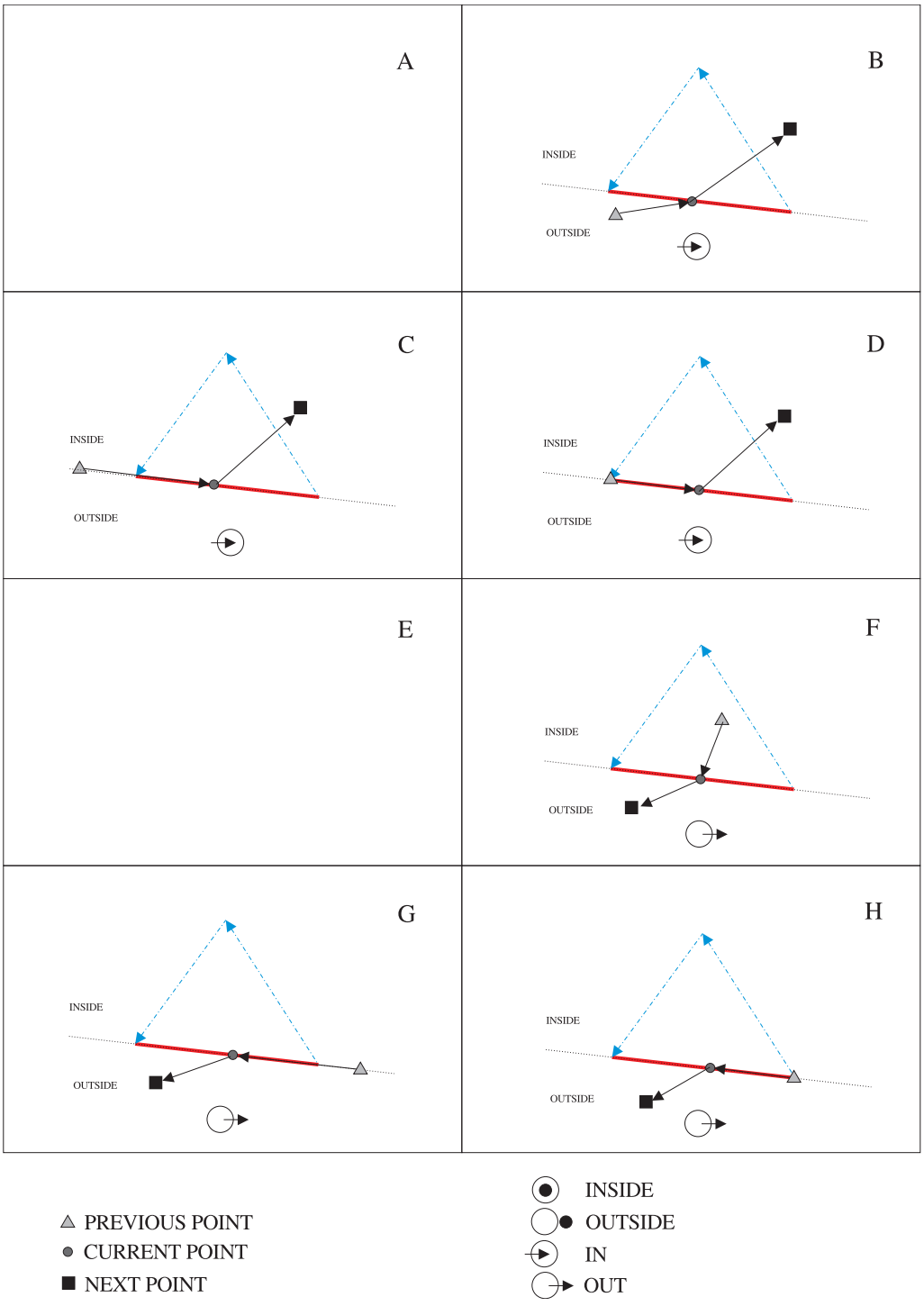

Figure 3.30: State changes in new intersections. Cases A and E do not produce changes.

Previous V.	Previous I.	Current V.	Next I.	Bext V.	State
OUTSIDE	NO	VERTEX_ALIG	NO	OUTSIDE	INSIDE-OUTSIDE
OUTSIDE	NO	VERTEX_ALIG	SI	OUTSIDE	IN
OUTSIDE	SI	VERTEX_ALIG	NO	OUTSIDE	OUT
OUTSIDE	SI	VERTEX_ALIG	SI	OUTSIDE	INSIDE
OUTSIDE	NO	VERTEX_ALIG	NO	INSIDE	IN
OUTSIDE	SI	VERTEX_ALIG	NO	INSIDE	INSIDE
INSIDE	NO	VERTEX_ALIG	NO	OUTSIDE	OUT
INSIDE	NO	VERTEX_ALIG	NO	DEMTRP	INSIDE

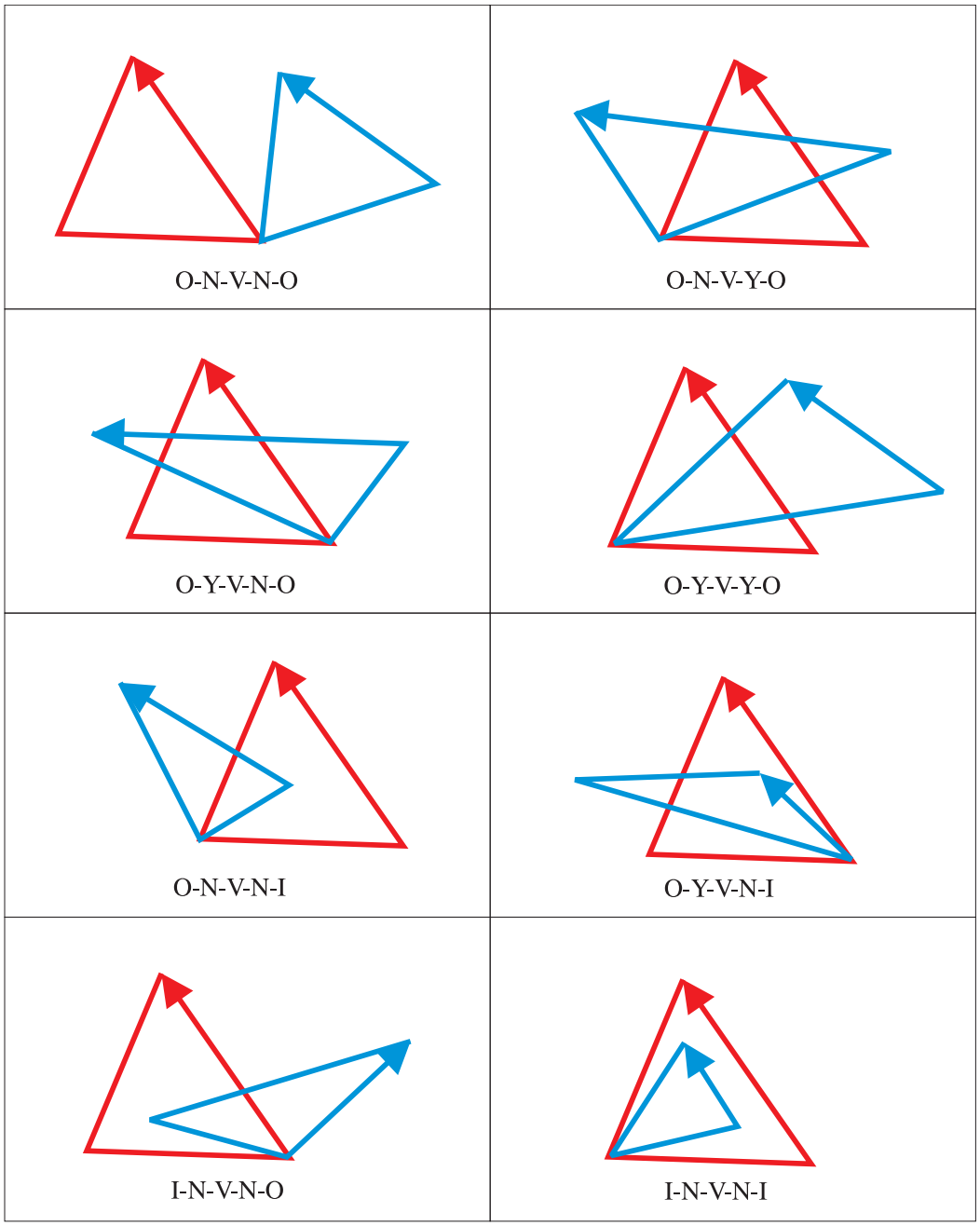

Figure 3.31: Visibility changes when vertices are coincident.

It is proved that only 8 possibilities of the 32 are valid. In Figure 3.31 can be seen graphically all the cases.

One of the results is INSIDE-OUTSIDE. The meaning is that the vertex position will be INSIDE or OUTSIDE depending upon the relative positions of the faces: if the triangle is ahead the state will be OUTSIDE while it will be INSIDE if the

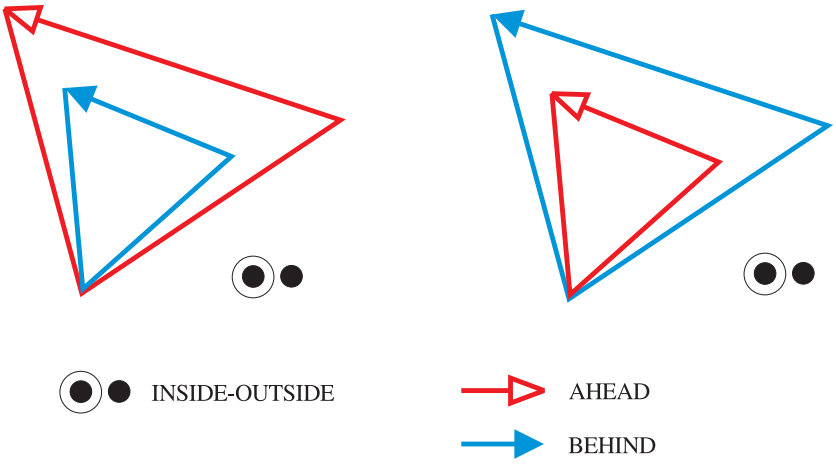

Figure 3.32: Case in which the vertex is INSIDE-OUTSIDE.

triangle is behind (Figure 3.32).

The new intersections are introduced when the state of the coincident vertices is determined. The update of vertices is applied only to those in which there is a visibility change (IN, OUT), or those that are inside (INSIDE). That vertices that are classified are OUTSIDE have no change.

Visualization

The visualization process is based in that the visibility changes only occur in intersections or because the vertices of one triangle are inside other one.

At first time both triangles are visible and vertices are marked as OUTSIDE/-AHEAD. When the processing of both triangles is finished, some new intersections

Figure 3.33: System structure.

can appear which will indicate if they come in or come out, ahead or behind. Maybe coincident vertices which have the same processing as the new intersections, and maybe constituent vertices that are inside the other triangle. These vertices are marked as INSIDE, and although they have not visibility change, they must be update because they are absolute references.

3.3 Animation

Non-Linear Transformations and our extension are representation independent, so it is not necessary to modify the method. The functions that have been implemented are B-splines, non rational and non uniform, as well as sine and cosine. They are simple but powerful.

The functions and data formats are detailed explained in Appendix A, section A.3.

3.4 System architecture

The system has been implemented using C++, and OpenGL for generating images. The structure of our model is shown (Figure 3.33). The normal pipeline is used to visualize objects in flat color or Gouraud shading. The second pipeline produces sketches. The control parameters are defined in script files.

OpenGL presents some problems with wide a very wide lines, even when the Offset mode is activated, because the wide of lines is orientation dependent.

3.5 Results

There have been developed some models to show the capabilities of the system for producing 2D animation and illustration. The models are simple because the system is not fully implemented, specially the user interface. We were interested in obtaining results more than to have a good interface. As commented previously, the control files are done by hand.

Even though the simplicity of the models, they have served as a test bed for our ideas. In Figure **??** different modes of visualization can be seen (A, wireframe; B, sketches; C, flat color; D, two-tone; E, Gouraud shading; F, shading plus specular brights). Some short animations including the different elements, sketches, virtual lights, and Hierarchical Extended Non-Linear Transformations have been produced. In Figure **??** there is an example of HENLT applied to a house character, doing a squash and stretch, at the same time that oscillates to both sides (there are not virtual lights). In Figure **??** there is another character, called “Brad Pig”, running (it has defined one diffuse and one specular virtual light). In Figures **??** and **??** several frames of an animation can be seen, the first one in flat color and the second one with Gouraud shading. it must be stand out that using a Pentium II, 300 Mhz, with 64 MB ram, with Linux and Mesa3D, but with no hardware acceleration and without optimizations, with a set of 22000 faces, as the model shown in Figures **??**

and **??**, there is an image generation that allows us to interact with the system. The next table shows times, for two resolutions and different number of images, with flat color. Time is shown is seconds and frames per second.

Flat color			
Res \ Ima	**50**	**100**	**200**
400x400	68/0.73	135/0.74	271/0.73
800x800	107/0.46	213/0.46	428/0.46

The next table is with the same data but with Gouraud shading.

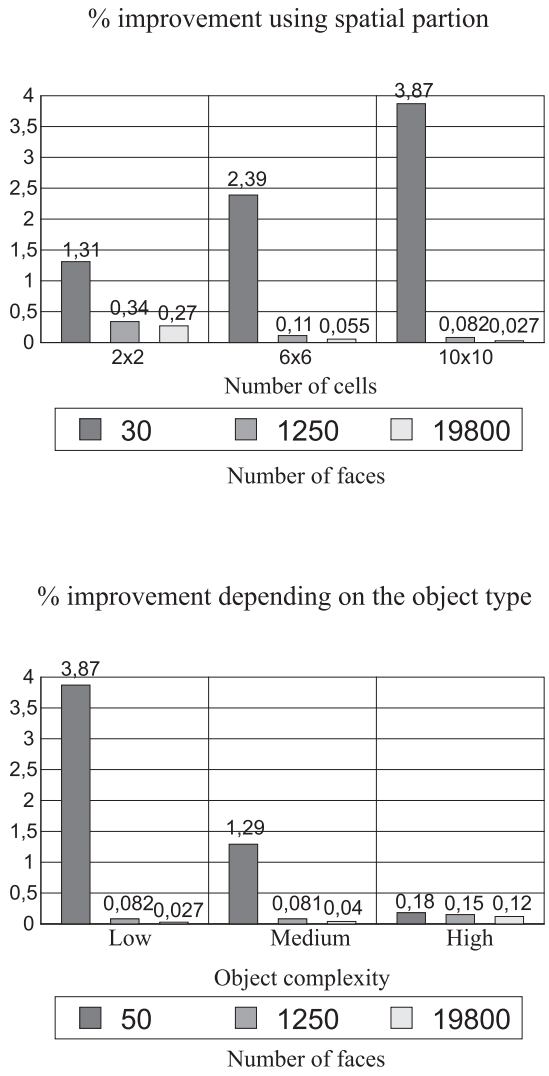

Figure 3.34: Results of applying spatial partition.

Gouraud shading			
Res \ Ima	**50**	**100**	**200**
400x400	71/0.70	143/0.69	286/0.69
800x800	111/0.45	222/0.45	444/0.45

These times include sketches processing and virtual lights, as well as the visualization, which is done almost totally in software. For the same scene excluding the visualization a time of 62 seconds for computing 100 frames has been taken. This time gives a ratio of 1,61 images per second. Due to the interrelation of the software elements it has not been possible to isolate the visualization computing from the rest. So, this time can be considered as a maximum.

In Figure **??** there is an example of deformations applied to a cylinder which simulates the leg of the pig. In Figure **??** there are several frames of an animation showing several objects with their own virtual lights. Finally, in Figures **??** and **??** there are frames of animations of two characters, "Poly Famous the martian" and "Roket E.". The capabilities of modelling and expressivity can be seen

3.5.1 Optimization study using spatial partition

As commented previously, the algorithm that computes the intersections works in normalized coordinates, and has quadratic complexity. One manner of reducing the number of operations is using a spatial partition. The idea is that intersections are spatially localized. The first possibility that we have tried is to divide the normalized coordinate system in a mesh. The triangles are located in cells of the mesh. Once the classification has been achieved, given one triangle we can know which cells it occupies and also which triangles it must be compared with.

This optimization possibility has been partially implemented, and the results have been very hopeful. In Figure 3.34 there are the obtained results. The showed data are relations between the number of comparisons that normally are done and the number of comparisons using spatial partition (value 1 indicates no improvement, less than 1 is better, and greater than 1 is worse). The first graphic show data about one object with several levels of detail (more or less faces) in relation with the mesh size. In the second graphic there are different objects (more or less complex), for a mesh size of 10x10 (the more complex object is a random surface). As can be seen, we have found improvements up to 98% in 10x10 cells meshes.

Chapter 4

Conclusions

The model *Virtual Lights* has been presented in this memory. Its implementation allows us to obtain one of the more important characteristics for obtaining images with a 2D appearance: sketches. A system that uses *Virtual Lights* for defining and obtaining internal and external sketches has been implemented. The method is simple and can be used with different geometric models.

Internal and external sketches have been defined for a general geometric model and for a polygonal geometric model. Though both type of sketches have different characteristic, they can be treated in a unified way thanks to the *Virtual Lights* model.

The detection and classification of sketches is done based in different visual characteristics which depend on virtual lights. Virtual lights not only allow to detect sketches but also that the user define them in an easy and intuitive way.

The intersections must be computed due to that maybe intersections between sketches. A new algorithm has been presented that can be also used for doing hidden line elimination, extending the Appel's *quantitative invisibility* concept, in the sense that also controls how many times and edge appears ahead of faces. Though currently our method is not as fast as Markosian's one, it presents interesting advantages: it is complete, it always find external sketches, allow to form chains that are not totally included in the vision volume, without breaking them. The quantitative visibility is a concept that adds flexibility in visualizing sketches.

The algorithm that computes intersections is based in doing the computing with two triangles each time, and adding the partial results.

The need of being able of transforming the solid models in such a way that have the same characteristics as in animation and natural beings simulation, carried out to find a method that allows those capabilities. Though the Free Form Deformation is widely used due to is capabilities, the Hierarchical Extended Non-Linear Transformation is proposed as alternative method that has a very economical representation, though a less localized control.

The obtained results with HENLT are good, and we consider that as opposed to the more localized control of FFD, the HENLT allow us to define deformations in a easy and economic way, for most of the transformations. The capability of combining constant and non-linear transformations as well as the use of hierarchies opens new

possibilities. The method can be used as a modelling tool.

The HENLT method allows to define static deformations. The control function can be changed in time. A method based in the use of functions that can reference other functions has been implemented. The mechanism is very simple but very powerful defining movements, under user control.

All these elements have been implemented in a system that allow us to show the capabilities and limitations. The results have been satisfactory, as can be seen in color images. The system can be used not only for doing animation but for illustrations and technical drawing.

The more important contributions are:

- Definition of sketches (Chapter 2, Page 25).
- Implementation of an algorithm for producing sketches with polygonal models (Chapter 3, Page 41).
- Creation and definition of the *Virtual Lights* model (Chapter 2, Page 27).
- Implementation of *Virtual Lights* model for polygonal models (Chapter 3, Page 47).
- A new algorithm of intersections computing between triangles, based in *quantitative visibility* (Chapter 3, Page 50).
- Definition of Hierarchical Extended Non-Lineal Transformations (Chapter 2, Page 31), extension of Non-Linear Transformations method, generalizing for using hierarchies.
- Development of animation by means of using functions that are applied to control functions of Hierarchical Extended Non-Linear Transformations. (Chapter 2, Page 37).
- Implementation of a system that allow us to create animations with a 2D appearance.

4.1 Future developments

The implemented system has been used to check the initial ideas, which has been reached. However, this is not final phase but it serves as a start point for other developments and improvements.

Some of the improvement possibilities are:

- Spatial partition
  Finishing which was commented in previous chapter.
- Temporal-spatial coherence
  The reusing of information in time is another possibility of optimizing the system, based in previous works [29, 42].

- Reduction of the number of faces
  This possibility joined to the above one, establishing several levels of detail, is very interesting. Some important works are [20, 22].

- Other improvements
  Other possibility is to improve the data sets that OpenGL uses, for example, it is faster when uses triangle strips instead of individual triangles. An example of work is [63].

  The improvement of the user interface of several modules: modeler, virtual lights, functions.

Once the improvements are commented, the future research lines are presented.

The first line is to develop a mixed 3D+2D space. OpenGL is limiting the possibilities of producing lines with "character". A space that mix the capabilities of 3D space, the use of models, illumination, virtual lights, and so on, and a 2D space that allow us to work with sketches without the problem produced by depth and orientation, can be the solution.

Another line is the development of HENLT. Currently we are researching for the possibility of obtaining a more localized control using spatial hierarchies or spatial partition.

Appendix A

Implementation of data formats

The data format of the several elements used in the system are presented in this appendix. There is also a example of Hierarchical Extended Non-linear Transformation for a better understanding of how works.

The information used for the system is structured in three files: models file, functions file, and camera file.

A.1 Models data format

The models file has information about every actor or character of the scene. The data that defines objects, attributes, local functions, and so on, are defined for every character.

The file format is:

Element_1
Element_2
⋮
Element_N

For every element there is the next format:

TIPO
NOMBRE
Geometry definition
Attributes
Constant Transformations
Virtual lights definition
Local functions definition
HENLT definition

The format for every component is:

- *TIPO:* SMANUAL | SBIPARAMETRICA | SBILINEAL | MONTANIA_FRACTAL | SBSPLINE | SREVOLUCION
  These are the different objects that can be modelled.

- *NOMBRE:*
  The name of the object

- *Geometry definition:*
  The data format for every type is shown below.

- *Attributes*
  The starting point of attributes is indicated with: * ATRIBUTOS. The parameters are:

  - ILUMINACION: SI | NO
    It indicates if the illumination model is or not be applied.
  - MODO_SUAVIZADO: SUAVIZADO | PLANO
    It indicates if there is or not Gouraud shading.
  - COLOR_RELLENO: (x, y, z) being $x, y, z \in R$ and $0 \leq x, y, z \leq 1$
    It indicates the fill color, if not illumination model is applied. The RGB color model is used.
  - MATERIAL: *ambient, diffuse, specular, exponent*
    The ambient, diffuse, specular and the exponent are indicated.
    ambient, diffuse, specular: (x, y, z) being $x, y, z \in R$ and $0 \leq x, y, z \leq 1$
    exponent: $\forall x \in R$ such that $0 \leq x \leq \infty$
  - OPACIDAD: x such that $x \in R$ and $0 \leq x \leq 1$
    It indicates the opacity level. 0 is transparent and 1 is totally opaque.
  - GROSOR_LINEA: *grosor_fino, grosor_grueso*
    It indicates the thickness of lines.
    grosor_fino, grosor_grueso: x such that $x \in N$ and $0 \leq x \leq MaxOGL$

- *Constant transformations*
  This transformations do not use the HENLT mechanism.

  - Translations
    TRASLACION:(δx, δy, δz)
  - Scaling
    ESCALADO:(S_x, S_y, S_z)
  - Rotations
    ROTACION_EJE_X: angle (grades)
    ROTACION_EJE_Y: angle (grades)
    ROTACION_EJE_Z: angle (grades)

- *Virtual lights definition*
  The starting point of virtual lights is indicated with: * LUCES. The data format of virtual lights is detailed presented in section A.2.

- *Local functions definition*
  The starting point of functions is indicated with: * FUNCIONES. The data format of functions is detailed presented in section A.3.

- *HENLT definition*
  The starting point of functions is indicated with: * TNLEJ. The data format of HENLT is detailed presented in section A.4.

The detailed data format of the different objects that can be defined is presented below.

- SMANUAL
  A manual surface is a mesh of mxn points, in a matrix arrangement. The vertex are manually defined.

  The format is:

  - NUM_PUNTOS_U: number of points
  - NUM_PUNTOS_V: number of points
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

- SBILINEAL
  A bilinear surface is define with four points. The number of divisions is indicated.

  The format is:

  - NUM_DIVISIONES_U: number of divisions
  - NUM_DIVISIONES_V: number of divisions
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

- SBIPARAMETRICA
  The ruled surface is defined with two curves, which are defined as B-splines.

  The format is:

  - NUM_PUNTOS_U: number of points
  - NUM_PUNTOS_V: number of points
  - NUM_DIVISIONES_U: number of divisions
  - ORDEN_U: order
  - TIPO_CURVA_EJE_U: UNIFORME | NO_UNIFORME
    It indicates if the curve is or not uniform.
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

- MONTANIA_FRACTAL
  This is a mountain-like surface created form a bilinear surface and changing the height in a random manner.

  The format is:

  - NUM_DIVISIONES_U: number of divisions
  - NUM_DIVISIONES_V: number of divisions
  - ALTURA: maximum height
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

- SBSPLINE
  This is a B-spline surface.

  The format is:

  - NUM_PUNTOS_U: number of points
  - NUM_PUNTOS_V: number of points
  - NUM_DIVISIONES_U: number of divisions
  - NUM_DIVISIONES_V: number of divisions
  - ORDEN_U: u order
  - ORDEN_V: v order
  - TIPO_CURVA_EJE_U: UNIFORME | NO_UNIFORME
    Type of u curve.
  - TIPO_CURVA_EJE_V: UNIFORME | NO_UNIFORME
    Type of v curve.
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

- SREVOLUCION
  This surface is generated by rotating a curve along an axis.

  The format is:

  - NUM_PUNTOS_U: $0 \mid x \neq 0$
  - NUM_PUNTOS_V: number of points
  - NUM_DIVISIONES_U: number of divisions
  - NUM_DIVISIONES_V: number of divisions
  - ORDEN_U: u order
  - ORDEN_V: v order
  - TIPO_CURVA_EJE_U: UNIFORME | NO_UNIFORME
    Type of u curve.
  - TIPO_CURVA_EJE_V: UNIFORME | NO_UNIFORME
    Type of v curve.
  - EJE_REVOLUCION: EJE_X | EJE_Y | EJE_Z
    It indicates along which axis the rotation is done
  - PUNTOS_DE_CONTROL: $(x, y, z), \ldots$

A.2 Virtual lights data format

The format of the virtual lights that are defined for each object is presented.

The format is:

type_of_light, coordinates, type_of_position, mobility

con:

- *type_of_light*: DIFUSA | SPECULAR
  The virtual light can have associated a diffuse or specular illumination model.

- *coordinates*: (x, y, z)
- *type_ of_ position*: POSICIONAL | DIRECCIONAL
  The virtual light can be at infinity (directional) or near the scene (positional).
- *movilidad*: SI | NO
  It indicates if the light is moved in relation to the object is associated. Generally, they are static.

Once the virtual lights are defined, we need to define the conditions that control their application. It is necessary to give the individual conditions for faces, the united condition for both faces, in relation to normal vectors, as well as reflections.

The format is:

face: value_ 1, value_ 2, condition, exponent

con:

- *face*: CARA1 | CARA2
  The conditions are define for both faces that share an edge.
- *value_ 1, value_ 2*: $\forall x \in R$
  Values of conditions.
- *condition*:
  LI=Lower limit
  LS=Upper limit
  operations: >, >=, <, <=, !=, =, NINGUNA
  It indicates the operation that will be applied and the conditions that normal vector have to carry out.
- *exponent*: $\forall x \geq 0 \in R$

The global operation is defined with:

The format is:

CONJUNTA: valueN_ 1, valueN_ 2, normals_ condition, relation, operation_ reflection, valueR_ 1, valueR_ 2, reflection_ condition

with:

- *valueN_ 1, valueN_ 2, valueR_ 1, valueR_ 2*: $\forall x \in R$
- *normals_ condition, reflection_ condition*:
  LI=Lower limit
  LS=Upper limit
  operations: > | >= | <| <= | != | =
- *relation*: CARA1_MENOR_CARA2 |
  CARA1_MAYOR_CARA2 |
  CARA1_IGUAL_CARA2 |
  INDISTINTA

- *reflection_ operation*: ABS(ABS(CARA1)+ABS(CARA2)) |
  ABS(ABS(CARA1)-ABS(CARA2)) |
  ABS(ABS(CARA2)-ABS(CARA1)) |
  ABS(CARA1+CARA2) | ABS(CARA1-CARA2) |
  ABS(CARA2-CARA1) | CARA1+CARA2 |
  CARA1-CARA2 | CARA2-CARA1

At first time the relations and conditions of normal vectors are used. Then, the reflections are used. They all must complies with the imposed conditions. The three operations that are shown in Figure 3.11 are defined here.

The global operation is applied finally.

The format is:

condition OPL condition OPL ...

con:

- *condition*: REFLEXION_SI | REFLEXION_NO
  It is indicated if the edge has been selected (REFLEXION_SI) or has not been selected (REFLEXION_NO) for every virtual light, taken by its position in the list.
- *OPL*: O | Y
  The allowed logical operations are: OR and AND.

A.3 Functions data format

As commented previously, the HENLT mechanism is based in functions. The format of the function is:

function_ number, function_ class, function_ type, abscises, ordinates, parameters

with:

- *function_ number*: function identifier.
- *function_ class*: FUNCION | FUNCION_D
  This variable indicates if the function is normal (FUNCION), or if its abscises are dependent of other function (FUNCION_D).
- *function_ type*: SENO | COSENO | BSPLINE | BSPLINE_X. It indicates which type of function will be used.
- *abscises*: function_number | $Min_{abscise}$, $Max_{abscise}$
  If the type of function is FUNCION_D, abscises are indicated by a function number (FUN_X, X is the function number), and if it is of type FUNCION, the abscises are given by values (MIN_X, MAX_X, MIN_Y, MAX_Y, MIN_Z, MAX_Z, are alias of the max value of the bounding box).
- *ordinates*: $Min_{ordinate}$, $Max_{ordinate}$

- *parameters*: Each type of function has their own definition parameters.

The format of each kind of function is:

- **SENO**
  The sine parameters are:

  amplitude, frequency, phase and shift.

  The cosine function is a sine function with 90 degrees out of phase. The sine function is defined between 0 y 2π by default. If the abscises have other values, a linear interpolation between that values and 0 and 2π is applied.

  Animation
  The time variation of a sine function is achieved changing some or all the parameters, which can depend on other function.

- **BSPLINE**
  The B-spline parameters are:

  order, number of points, list of 3D points (x, y, z).

  The abscises are defined by the parameter range, while the ordinates are y values of the computed point of the curve. If the abscises that the user wants are not the same that the default ones, there is a linear interpolation. The limit value of the ordinates are define by the max values of the bounding box of the curve. If they do not coincide with the user's ones, there is a linear interpolation.

  Animation
  The animation is obtained making the limit values or the ordinates can change, depending of other functions (Figure 2.15). Due to the limitation that this function presents, we implemented an extension: BSPLINE_X.

- **BSPLINE_X**
  The bspline_x parameters are:

  order, number of points, list of 3D point.

  The great difference is that points that define the control polygon of the B-spline could not be static but can be defined as:

  $(x, y = f(s), z)$.

  This indicates that y coordinate depends upon a function. One characteristic is that the animation surface could not be a B-spline one, because each function have not to be a B-spline. This function is very flexible and powerful defining animated transformations. The rest of parameters are similar to the BSPLINE ones.

Now we present an example in which the sine function must return a value between 0 and 360 degrees when the user pass to the function a value between 0 and 1:

$\alpha = f(sine(\beta);;0, 360)$ y $\beta = f_{basica}(y = x; 0, 1; 0, 2\pi)$

The recursive dependence has a defined structure: Deformation_functions → animation_functions → master-clock_function. The system uses a procedure called

master-clock which controls what can be called "global time", which is used for the rest of functions. This procedure control how many "ticks" or frames are produced.

The main procedure has the next structure:

```
void Main()
{
while (masterClock.tick())
  {
  ObtainImages
  }
}
```

A.4 HENLT data format

The HENLT are based in functions, as commented previously.

The format is:

transformation_ class, actualization, transformation, parameters.

con:

- *transformation_ class*: CONSTANTE | VARIABLE
  If the value is CONSTANTE the transformation is constant, that is, it is the same for all the geometry, but it can change in time. If the value is VARIABLE it is a HENLT.
- *actualization*: SI | NO
  It indicates if the values of the initial list must be update with the values of the final list.
- *transformation*: TRASLACION | ESCALADO | ROTACION
- *parameters*:
  The parameters depend upon the class of transformation.
  - TRASLACION:
    - CONSTANTE:
      δx | FUN_X, δy | FUN_X, δz | FUN_X;
      The parameters for a translation are the shifts that must be applied to each coordinate, or a number of a local function.
    - VARIABLE:
      selection axis, *application axis*, *number of local function*;
      selection axis, *application axis*: EJE_X | EJE_Y | EJE_Z
  - ESCALADO:
    - CONSTANTE:
      S_x | FUN_X, S_y | FUN_X, S_z | FUN_X;
      The parameters for a scaling are the scale factors that can be fixed or a number of a local function.

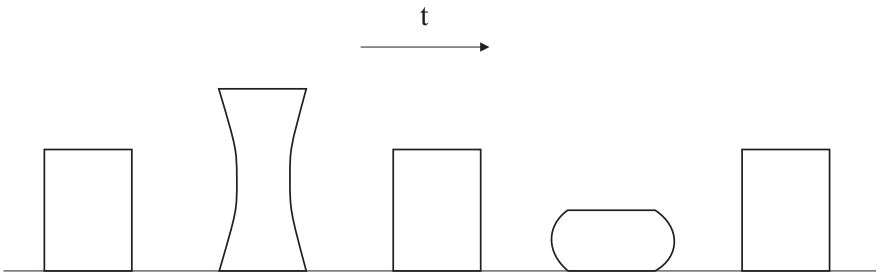

Figure A.1: Squash and stretch example.

  - ∗ VARIABLE:
    selection axis, *application axis*, *number of local function*;
    selection axis, *application axis*: EJE_X | EJE_Y | EJE_Z
- ROTACION:
  - ∗ CONSTANTE:
    application axis, *angle* / *FUN_X*;
    it indicates the rotation angle (degree) with a fixed value or a number of a local function.
  - ∗ VARIABLE:
    selection axis, *application axis*, *number of local function*;
    selection axis, *application axis*: EJE_X | EJE_Y | EJE_Z

A.4.1 Hierarchical Extended Non-Linear Transformation example

As commented previously, the composition of transformations is done in a logical way, because there is not a easy manner of obtaining only one transformation that combines the effect of several non-linear transformations. The composition is obtained as an accumulative process. We will show this with an example which implements the classical animation principle of squashing and stretching of a cube (Figure A.1). The first thing is to define a function that allows the cube goes up and down in a linear way along its y Axis. The cube has a height 100, and we want that it maximum extension is 150 and the maximum contraction is 50 (Figure A.2). The deformation is implemented as a translation, so we have to add 50 for extending and subtract 50 for contracting. The last step is to define how to pass from one temporal position to the other. It is possible to use a linear function (Figure A.3, A), but we have selected a sine function with an amplitude of 50 (Figure A.3, b). The other point remains constant. Now we show the code for this transformation (the master-clock is defined, global function 0).

Sine definition:

```
1:FUNCION_D:SENO,FUN_0,50,1,0,0;
```

Now we define the deformation control function, which is achieved with a local B-spline function:

```
0:FUNCION:BSPLINE,MIN_Y,MAX_Y,0,FUN_1,2,2,(0,0,0),(1,1,0);
```

and finally the HENLT:

```
VARIABLE,NO,TRASLACION,EJE_Y,EJE_Y,0;
```

If we want to add that the volume is maintained, a transformation has to be added for obtaining that effect. The form of the curve, given the original object, is shown in Figure A.4. It can be seen that the definition is very intuitive. The transformation will be done with a scaling: when the object blow up the scale factor is greater than 1 and when the object is deflated the scale factor is less than 1. The transformation is implemented changing the limits of the ordinates, while the curve form is maintained (Figure **??**).

The global function is defined as:

```
2:FUNCION_D:SENO,FUN_0,-.4,10,0,1;
```

There is shift that makes that the minimum and maximum values can be $1+.4 = 1.4$ y $1-.4 = .6$. The local function that define the deformation is:

```
1:FUNCION:BSPLINE,MIN\_Y,MAX\_Y,1,FUN\_2,3,3,(0,0,0),
                                    (0.5,1,0),(1,0,0);
```

The form of the curve is fixed but the limits change (Figure A.5). Finally, the HENLT is defined. As we want the transformation affects all the object, we have to apply the same transformation two time with the same selection axis but with two different application axes.

```
VARIABLE,NO,TRASLACION,EJE_Y,EJE_Y,0;
VARIABLE,NO,ESCALADO,EJE_Y,EJE_Z,1;
VARIABLE,NO,ESCALADO,EJE_Y,EJE_X,1;
```

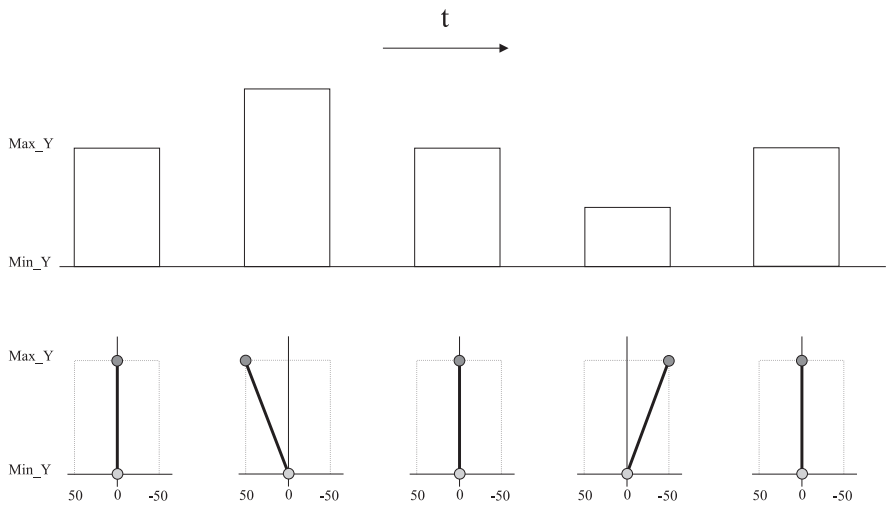

Figure A.2: Extension and contraction with a translation.

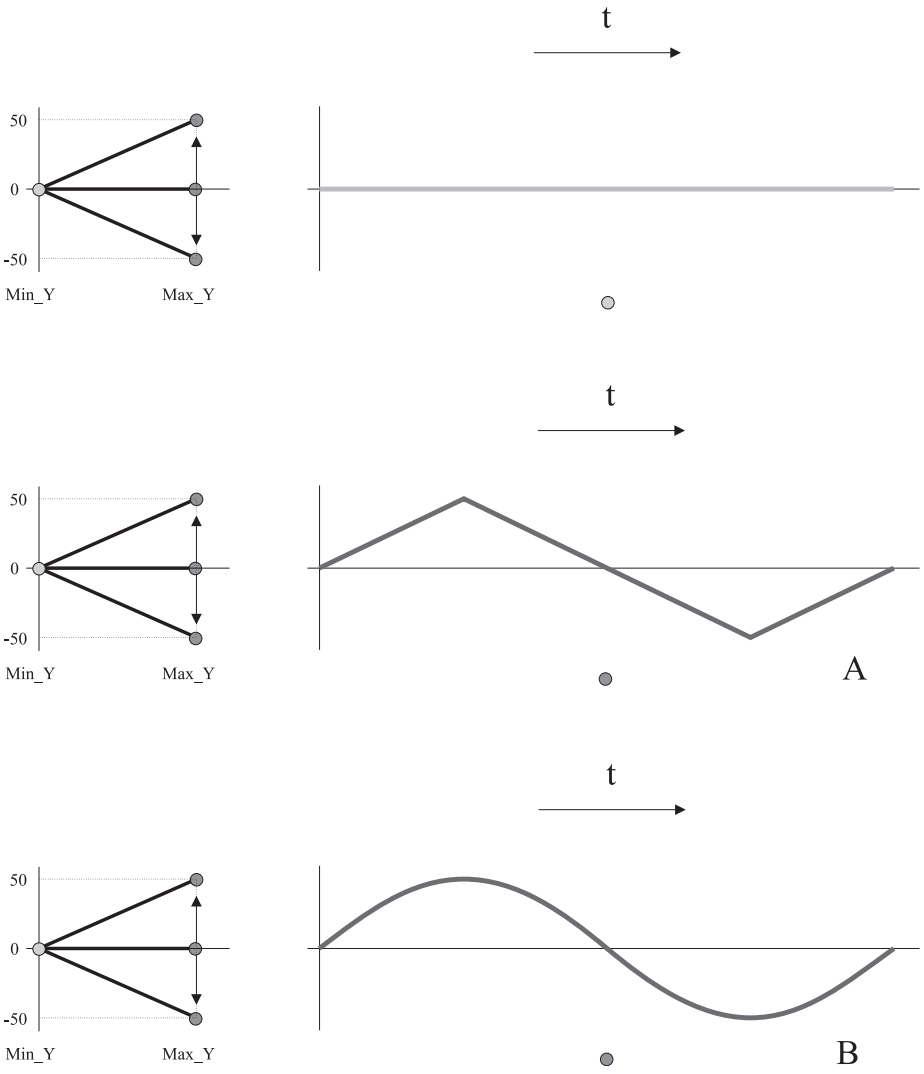

Figure A.3: Temporal dependence of translation control function.

At last, we add the oscillating movement. The definition is:

Global functions:

```
1:FUNCION_D:SENO,FUN_0,50,1,0,0;
2:FUNCION_D:SENO,FUN_0,-.4,10,0,1;
3:FUNCION_D:SENO,FUN_0,30,5,0,0;
```

Local functions:

```
0: FUNCION: BSPLINE, MIN_Y, MAX_Y, 0, FUN_1, 2, 2,
                                  (0,0,0), (1,1,0);
1: FUNCION: BSPLINE, MIN_Y, MAX_Y, 1, FUN_2, 3, 3,
                       (0,0,0), (0.5,1,0), (1,0,0);
2: FUNCION: BSPLINE, MIN_Y, MAX_Y, 0, FUN_3, 2, 2,
                                  (0,0,0), (1,1,0);
```

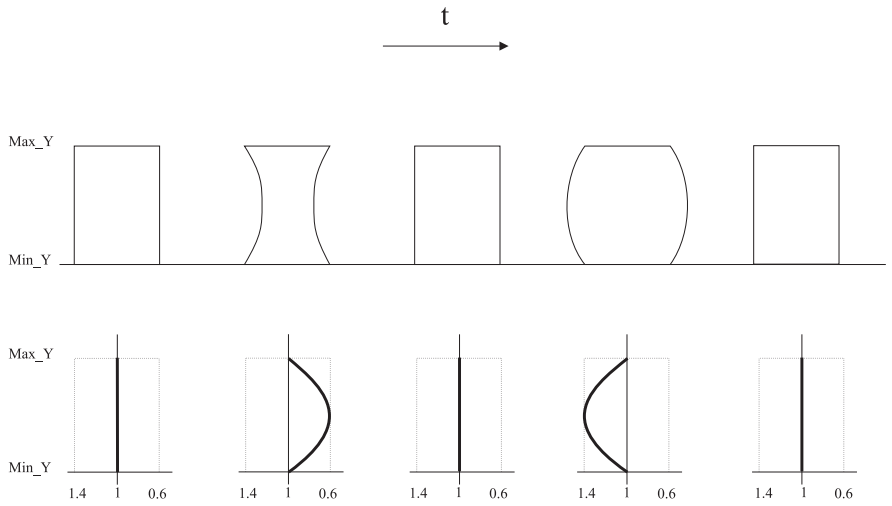

Figure A.4: Definition of scaling control function.

HENLT:

```
VARIABLE,NO,TRASLACION,EJE_Y,EJE_Y,0;
VARIABLE,NO,ESCALADO,EJE_Y,EJE_Z,1;
VARIABLE,NO,ESCALADO,EJE_Y,EJE_X,1;
VARIABLE,NO,ROTACION,EJE_Y,EJE_X,2;
```

A.5 Camera data format

This section shows the capabilities and data forma of the camera.

The general format is:

Number of frames
Parameters for locating the observer (PHIGS-like)
Parameters of projection and visualization
Vision attributes

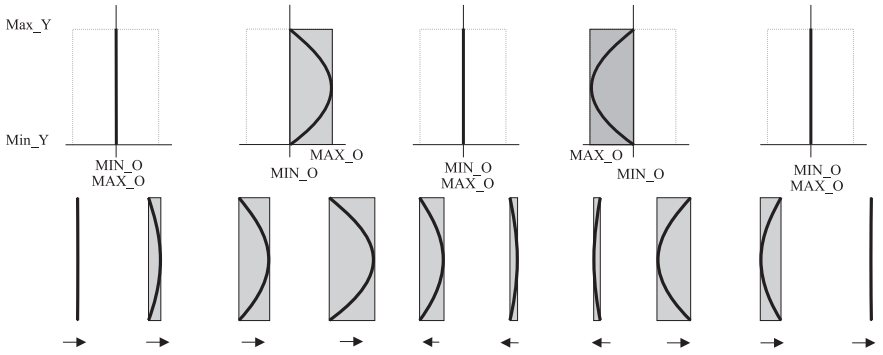

Figure A.5: Variation of ordinates.

Axis attributes
Camera movement control functions
Real lights definition and attributes

The concrete format is:

- *Number of images*
  - NUM_PASOS:
    It indicates the number of images that must be obtained.
  - INICIO:
    It indicate the start position.
  - FINAL:
    It indicates the stop position.
  - REPETICION: SI | NO
    It indicates if there is or not a loop.
- *Parameters for locating the observer*
  - VRP: x, y, z
  - VPN: x, y, z
  - VUP: x, y, z
  - PRP: x, y, z
- *Parameters of projection and visualization*
  - VENTANA_MUNDO: u_{min}, v_{min}, u_{max}, v_{max}
  - TIPO_PROYECCION: PARALELA | PERSPECTIVA
  - PLANOS_DE_CORTE: front plane, back plane
  - PUERTO_VISION: x_{min}, y_{min}, x_{max}, y_{max}
    Left handed coordinate system.
  - POS_VENTANA: x_{orig}, y_{orig}
    Window position in Xwindows.
  - TAM_VENTANA:x_{tam}, y_{tam}
    Window size in Xwindows.
- *Vision attributes*
  - DOBLE_BUFER: SI | NO
  - COLOR_FONDO: x, y, z being $x, y, z \in R$ y $0 \leq x, y, z \leq 1$
    Background color. RGB color model.
  - MODELO_ILUMINACION: ambient, diffuse, specular, specular exponent
    Parameter for OpenGL.
  - SUAVIZADO: SUAVIZADO | PLANO
    It indicates if there is or not shading using OpenGL.

  - PARAR: SI | NO
    It indicates if the program stops at each step.
  - LUCES: SI | NO
    It indicates if the lights positions are shown.
  - BITONO: SI |NO
    It indicates if the two-tone is active or not.

- *Axis attributes*
  - EJES: SI | NO
    It indicate is the axis are visualized or not.
  - TAM_EJES:
    it indicate the axis size.

- *Camera movements control functions*
  A control function is defined for each coordinate.

- *Definition of virtual lights and their attributes*
  - NUM_LUCES:
    Number of real lights.
  - POSICION_LUZ: x, y, z
  - COLOR_LUZ: x, y, z being $x, y, z \in R$ y $0 \leq x, y, z \leq 1$
    Light color.
  - TIPO_LUZ: POSICIONAL | DIRECCIONAL
  - LUZ_MOVIL: SI | NO
    It indicates if the light follows or not the camera.

A.6 Code example

As follows, there is the code that allows us to obtain the animation of Figure **??**.

Objects file (only two objects are shown).

```
[
#comentario
TIPO: SBSPLINE
NOMBRE: montania1
NUM_PUNTOS_U:5
NUM_PUNTOS_V:5
NUM_DIVISIONES_U:30
NUM_DIVISIONES_V:30
ORDEN_U:3
ORDEN_V:3
TIPO_CURVA_EJE_U:NO_UNIFORME
TIPO_CURVA_EJE_V:NO_UNIFORME
PUNTOS_DE_CONTROL:
(-1000,0,-1000), (-500,0,-1000), (0,0,-1000), (500,0,-1000), (1000,0,-1000),
(-1500,0,-500), (-500 1000,-500), (0,0,500), (500,800,-500), (1000,0,-500),
(-1500,0,-250), (-500,50, 250), (0,0,0),(500,100,-250), (1000,0,-250),
(-1500,0,500), (-500,0,500), (0,0,500), (500,0,500), (1000,0,500),
(-1000,-200,1000), (-500,-200,1000), (0,-200,1000), (500,-200,1000),
```

```
(1000,-200,1000)
#comentario
*  ATRIBUTOS
ILUMINACION:NO
MODO_SUAVIZADO:SUAVIZADO
COLOR_RELLENO:(0.2,.6,0.2)
MATERIAL:(.2,.8,.2),(.2,.8,.2),(1,1,1),10
OPACIDAD:0.2
GROSOR_LINEA:2,5
* TGNL
# tipo, eje de eleccion, eje aplicacion,numero de funcion
NUM_TGNL:3;
CONSTANTE,NO,TRASLACION,0,-80,-500;
CONSTANTE,NO,ROTACION,EJE_Y,45;
CONSTANTE,NO,ESCALADO,3,3,3;
]
[
TIPO:SREVOLUCION
NOMBRE:cuerpo1
# si numpuntosU=0 no se ajusta bspline
NUM_PUNTOS_U:1
NUM_PUNTOS_V:8
NUM_DIVISIONES_U:50
NUM_DIVISIONES_V:50
ORDEN_U:3
ORDEN_V:3
TIPO_CURVA_EJE_U:UNIFORME
TIPO_CURVA_EJE_V:NO_UNIFORME
EJE_REVOLUCION:EJE_X
PUNTOS_DE_CONTROL:(-800,0,0),(-800,0,800),(800,0,800),(800,0,230),(830,0,200),
                                                (1000,0,200),(1000,0,200),(1000,0,0)
* ATRIBUTOS
ILUMINACION:NO
MODO_SUAVIZADO:SUAVIZADO
COLOR_RELLENO:(1,.8,.8)
MATERIAL:(1,.8,.8),(1,.8,.8),(0,0,0),10
OPACIDAD:0.2
GROSOR_LINEA:1,3
* LUCES
NUM_LUCES_VIRTUALES:2
# tipo (DIF/ESP),pos (x,y,z),clase (DIR/POS),intensidad,movil
DIFUSA,1500,0,0,DIRECCIONAL,1,NO
ESPECULAR,1500,0,0,DIRECCIONAL,1,NO
* REFLEXIONES
# lim. inferior,lim. superior, operacion, condicion, exponente
CARA1:-1,1,>=LI_<=LS,1
CARA2:-1,1,>=LI_<=LS,1
CONJUNTA:0.5,1,>=LI_<=LS,CARA1_MAYOR_CARA2,ABS(CARA1-CARA2),0.5,1,>=LI_<=LS
CARA1:0.7,1,>=LI_<=LS,1
CARA2:0.7,1,>=LI_<=LS,1
CONJUNTA:0,0.95,>=LI_<=LS,CARA1_MENOR_CARA2,ABS(CARA1-CARA2),0.2,0.5,>=LI_<=LS
* OPERACION_LUCES
REFLEXION_SI O REFLEXION_SI
* FUNCIONES
# numero,tipo,absMin,absMax,ordMin (fun para funciones),ordMax,parametros
NUM_FUNCIONES:2
FUNCION_D:BSPLINE,FUN_0,0,720,2,2,(0,0,0),(1,1,0);
FUNCION_D:SENO,FUN_0,50,20,0,50;
* TGNL
#tipo con o var, tipo, eje de eleccion, eje aplicacion,numero de funcion PROPIA
NUM_TGNL:4;
```

```
CONSTANTE,NO,ESCALADO,0.3,0.3,0.3;
CONSTANTE,NO,TRASLACION,0,275,900;
CONSTANTE,NO,ROTACION,EJE_Y,FUN_0;
CONSTANTE,NO,TRASLACION,0,FUN_1,0;
]
```

Global functions file.

```
#numero de funcion,tipo, parametros
# funciones basicas constantes,
# numero,tipo,ordenadas,parametros ; se numeran apartir de 1 (0=reloj)
NUM_FUNCIONES:10;
FUNCION_D:BSPLINE,FUN_0,0,0,2,2,(0,0,0),(1,1,0);
FUNCION_D:BSPLINE,FUN_0,1,1,2,2,(0,0,0),(1,1,0);
FUNCION_D:BSPLINE,FUN_0,0,45,2,2,(0,0,0),(1,1,0);
FUNCION_D:SENO,FUN_0,30,5,0,0;
FUNCION_D:BSPLINE,FUN_0,0,50,3,3,(0,0,0),(0.5,1,0),(1,0,0);
FUNCION_D:SENO,FUN_0,200,10,0,0;
FUNCION_D:SENO,FUN_0,-.4,10,0,1;
FUNCION_D:SENO,FUN_0,200,10,0,800;
# para inflar el ojo
FUNCION_D:BSPLINE,FUN_0,0,200,2,3,(0,0,0),(0.5,1,0),(1,0,0);
# para inflar la pupila
FUNCION_D:BSPLINE,FUN_0,0,100,2,3,(0,0,0),(0.5,1,0),(1,0,0);
```

Camera file.

```
# reloj maestro,numero pasos,inicio,final,repeticion
NUM_PASOS:50
INICIO:0
FINAL:1
REPETICION:NO
# parametros de la camara
VRP:0,0,2000
VPN:0,0,1
VUP:0,1,0
PRP:0,0,1000
# umin,vmin,umax,vmax
VENTANA_MUNDO:-500,-300,500,700
TIPO_PROYECCION:PERSPECTIVA
# delantero, trasero
PLANOS_DE_CORTE:0,-8000
# xmin,ymin,xmax,ymax
PUERTO_VISION:0,0,800,800
POS_VENTANA:50,50
TAM_VENTANA:800,800
DOBLE_BUFER:SI
COLOR_FONDO:.8,.9,1
MODELO_ILUMINACION:.5,.5,.5,1
SUAVIZADO:SUAVIZADO
PARAR:NO
LUCES:NO
BITONO:NO
EJES:NO
TAM_EJES:2000
# se meten funciones-> ordmin,ordmax,orden,numpuntos,puntos
FUNCION_D:BSPLINE,FUN_0,2300,1300,2,2,(0,0,0),(1,1,0);
FUNCION_D:BSPLINE,FUN_0,700,300,2,2,(0,0,0),(1,1,0);
```

```
FUNCION_D:BSPLINE,FUN_0,2300,1300,2,2,(0,0,0),(1,1,0);
#FUNCION_D:COSENO,FUN_0,1500,1,0,0;
#FUNCION_D:BSPLINE,FUN_0,300,300,2,3,(0,0,0),(.5,1,0),(1,0,0);
#FUNCION_D:SENO,FUN_0,1500,1,0,0;
# luces reales
NUM_LUCES:1
POSICION_LUZ:1500,500,0
COLOR_LUZ:.5,0.5,0.5
TIPO_LUZ:DIRECCIONAL
LUZ_MOVIL:SI
```

Bibliography

[1] A. Barr. Global and local deformations of solid primitives. *Proceedings of SIGGRAPH*, 18(3):21–30, 1984.

[2] A. Barr, B. Currin, S. Gabriel, and J. Hughes. Smooth interpolation of orientations with angular velocity constraints. *ACM Computer Graphics*, 26(2):313–320, July 1992.

[3] A. M. Barr. Supercuadrics and angle preserving transformations. *IEEE Computer Graphics And Applications*, 1(1):11–23, 1981.

[4] T. Beier and S. Neely. Feature-based image metamorphosis. *ACM Computer Graphics*, 26(2):35–42, July 1992.

[5] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on graphics*, 1(3):235–256, 1982.

[6] L. Brotman and A. Netravali. Motion interpolation by optimal control. *ACM Computer Graphics*, 22(4):309–316, August 1988.

[7] N. Burntnyk and M. Wein. Interactive squeleton techniques for enhancing motion dynamics in keyframe animation. *Communications of the ACM*, 19(10):564–569, October 1976.

[8] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *ACM Computer Graphics*, 26(2):99–104, July 1992.

[9] E. Catmull. The problems of computer-assisted animation. *ACM Computer Graphics*, pages 348–353, 1978.

[10] D. Chen and D. Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. *ACM Computer Graphics*, 26(2):89–98, July 1992.

[11] P. H. Christensen. Contour rendering. *Computer Graphics*, 33(1):58–60, February 1999.

[12] S. Coquillart. Extended free form deformation: A sculpturing tool for 3D geometric modeling. *ACM Computer Graphics*, 24(4):187–196, August 1990.

[13] S. Coquillart and P. Jancene. Animated free form deformation: An interactive animation technique. *ACM Computer Graphics*, 25(4):23–26, July 1991.

[14] A. Daldegan, N. Magnenat-Thalmann, T. Kurihara, and D. Thalmann. An integrated system for modeling, animating, and rendering hair. *Computer Graphics Forum*, 12(3):211–221, September 1993.

[15] S. Donikiam and G. Hegron. A declarative design method for 3D scene sketch modeling. *Computer Graphics Forum*, 12(3):223–236, September 1993.

[16] D. Ebert and R. Parent. Rendering and animation of gaseous phenomena by commbining fast volume and scan-line a-buffer techniques. *ACM Computer Graphics*, 24(4):357–366, August 1990.

[17] G. Elber and E. Cohen. Hidden curve removal from free form surfaces. *ACM Computer Graphics*, 24(4):95–104, August 1990.

[18] G. Farin. *Curves And Surfaces For CAGD: A Practical Guide. 3 Edition.* Academic Press, 1993.

[19] J. D. Fekete et al. Tictactoon: A paperless system for professional 2D animation. *Proceedings of SIGGRAPH*, pages 79–90, August 1995.

[20] A. Finkelstein and D. H. Salesin. Multiresolution curves. *Proceedings of SIGGRAPH*, pages 261–268, July 1994.

[21] J. Foley, A. van Dam, S. Feiner, and J. F. Hughes. *Computer Graphics: Principles And Practice, 2 Edition.* Addison-Wesley, 1992.

[22] D. R. Forsey and R. H. Bartels. Hierarchical b-spline refinement. *ACM Computer Graphics*, 22(4):205–212, August 1988.

[23] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. *Proceedings of SIGGRAPH*, pages 447–452, July 1998.

[24] J. Gouret, N. Magnenat-Thalmann, and D. Thalmann. A non-photorealistic lighting model for automatic technical illustration. *Proceedings of SIGGRAPH*, pages 447–452, July 1998.

[25] S. Green. Beyond photorealism. *Workshop on Rendering*, July 1999.

[26] S. Greenberg. Why non-photorealistic rendering ? *Computer Graphics Forum*, 33(1):56,57, February 1999.

[27] P. Haeberli. Paint by numbers: Abstract image representation. *ACM Computer Graphics*, 24(4):207–214, August 1990.

[28] P. Hall. Non-photorealistic rendering by q-mapping. *Computer Graphics Forum*, 18(1):41–56, March 1999.

[29] H. Hoppe. View-dependent refinement of progressive meshes. *Proceedings of SIGGRAPH*, pages 189–198, August 1997.

[30] S. C. Hsu and I. H. Lee. Drawing and animation using skeletal strokes. *Proceedings of SIGGRAPH*, pages 109–118, July 1994.

[31] W. Hsu, J. Hughes, and H. Kaufman. Direct manipulation of free form deformations. *ACM Computer Graphics*, 26(2):177–184, July 1992.

[32] P. Isaacs and M. Cohen. Controling dynamics simulation with kinematics constraints, behavior functions and inverse dynamics. *ACM Computer Graphics*, 21(4):215–224, July 1987.

[33] P. Kalra, A. Mangili, N. Magnenat-Thalmann, and D. Thalmann. Simulation of facial muscle actions based on rational free form deformations. *Computer Graphics Forum*, 11(3):59–69, September 1992.

[34] S. Karan and E. Fiume. Wires: A geometric deformation technique. *Proceedings of SIGGRAPH*, pages 405–414, 1998.

[35] W. Kong and M. Nakajima. Visible volume buffer for efficient hair expression and shadow generation. *Computer Animation 99*, pages 58–65, 1999.

[36] D. Kromker and G. R. Hofmann. Interaction, integration, visualisation. Technical report, Eurographics, 1989.

[37] J. Lansdown and S. Schofield. Expressive rendering: A review of non-photorealistic techniques. *IEEE Computer Graphics and Applications*, pages 29–37, May 1995.

[38] J. Lasseter. Principles of traditional animation applied to 3D computer animation. *Proceedings of SIGGRAPH*, 21(4):35–44, July 1987.

[39] M. J. Lazlo. *Computational Geometry And Computer Graphics In C++*. Prentice-Hall, 1996.

[40] W. Leister. Computer generated copper plates. *Computer Graphics Forum*, 13(1):69–77, 1994.

[41] P. C. Litwinowicz. Inkwell: A $2\frac{1}{2}$d animation system. *Proceedings of SIGGRAPH*, 25(4):113–122, July 1991.

[42] D. Luebke and C. Erikson. View-dependent simplification of arbitray polygonal enviroments. *Proceedings of SIGGRAPH*, pages 199–208, August 1997.

[43] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. *Proceedings of SIGGRAPH*, pages 181–188, August 1996.

[44] N. Magnenat-Thalmann and D. Thalmann. *Computer Animation: Theory And Practice. Second Edition.* Springer Verlag, 1990.

[45] L. Markosian et al. Real-time non-photorealistic rendering. *Proceedings of SIGGRAPH*, pages 415–419, 1997.

[46] D. Martín. Revisión de métodos para obtener siluetas. *Jornadas de Informática Gráfica*, pages 61–70, September 1998.

[47] D. Martín and J. C. Torres. Obtención y visualización rápida de trazos. *8 Congreso Español de Informática Gráfica CEIG98*, pages 95–108, 1998.

[48] D. Martín and J. C. Torres. *Alhambra*: A system for producing 2D animation. *Computer Animation 99*, pages 38–47, 1999.

[49] D. Martín and J. C. Torres. *Virtual Lights*: A method for expressive visualisation. *EG99-SP*, pages 67–70, September 1999.

[50] D. Martín, J. C. Torres, and V. del Sol. A model for production of two dimensional animation. *Fourth Eurographics Animation and Simulation Workshop*, pages 11–22, September 1993.

[51] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. *ACM Computer Graphics*, 26(2):309–312, July 1992.

[52] M. E. Mortenson. *Geometric Modeling.* John Wiley & Sons, 1985.

[53] S. Muraki. Volumetric shape description of range data using "blobby model". *ACM Computer Graphics*, 25(4):227–235, July 1991.

[54] H. Nishimura et al. Object modelling by distribution function and a method of image generation. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, J68-D(4):718–725, July 1985.

[55] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *ACM Computer Graphics*, 23(3):215–222, July 1989.

[56] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness. Animation of plant development. *Proceedings of SIGGRAPH*, 27(4):351–360, July 1993.

[57] R. Richens and S. Schofied. Interactive computer rendering. *Architectural Research Quarterly*, 1(1), 1995.

[58] P. Rustagi. Silhouette line display from shaded models. *Iris Universe*, pages 42–44, Fall 1989.

[59] T. Saito and T. Takahashi. Comprehensible rendering of 3D shapes. *ACM Computer Graphics*, 24(4):197–206, August 1990.

[60] M. P. Salisbury, C. Anderson, D. Lischinski, and D. H. Salesin. Scale-dependent reproduction of pen-and-ink illustration. *Proceedings of SIGGRAPH*, pages 461–468, August 1996.

[61] M. P. Salisbury, S. E. Anderson, R. Barze, and D. H. Salesin. Interactive pen and ink illustration. *Proceedings of SIGGRAPH*, pages 101–108, July 1994.

[62] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen and ink illustration. *Proceedings of SIGGRAPH*, pages 401–406, August 1997.

[63] J. Santoja and S. Bayarri. Resolución del problema de extracción de tiras triangulares mediante algoritmos genéticos. *8 Congreso Español de Informática Gráfica CEIG98*, pages 149–162, 1998.

[64] T. Sederberg and E. Greenwood. A physically based approach to 2-d shape blending. *ACM Computer Graphics*, 26(2):25–34, July 1992.

[65] T. Sederberg and S. R. Parry. Free form deformation of solid geometric models. *Proceedings of SIGGRAPH 86*, 20(4):151–160, 1986.

[66] T. W. Sederberg and A. K. Zundel. Scan line display of algebraic surfaces. *ACM Computer Graphics*, 23(3):147–156, July 1989.

[67] D. D. Seligman and S. Feiner. Automated generation of intent-based 3D illustration. *ACM Computer Graphics*, 25(4):123–132, July 1991.

[68] M. Shinya and A. Fourier. Stochastic motion-motion under the influence of wind. *Computer Graphics Forum*, 11(3):119–128, September 1992.

[69] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. *Proceedings of SIGGRAPH*, 27(4):279–280, July 1993.

[70] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *ACM Computer Graphics*, 22(4):269–278, August 1988.

[71] F. Thomas and O. Johnston. *The Illusion Of Life: Disney Animation.* Hiperion, 1995.

[72] K. Waters. A muscle model for animating three dimensional facial expressions. *ACM Computer Graphics*, 19(3):17–24, July 1985.

[73] J. Wejchert and D. Haumann. Animation aerodynamics. *ACM Computer Graphics*, 25(4):19–22, July 1991.

[74] T. White. *The Animator's Workbook. 3 Edition.* Phaidon-Press Limited, 1992.

[75] G. Winkenbach and D. H. Salesin. Computer generated pen and ink illustration. *Proceedings of SIGGRAPH*, pages 469–476, July 1994.

[76] G. Winkenbach and D. H. Salesin. Rendering parametric surfaces in pen and ink. *Proceedings of SIGGRAPH*, pages 469–476, August 1996.

[77] Y. Wu, P. Beylot, and N. Magnenat-Thalmann. Skin agin estimation by facial simulation. *Computer Animation 99*, pages 210–219, 1999.

[78] G. Wyvill, C. McPheeters, and B. Wywill. Data structure for soft objects. *Visual Computer*, 2(4):227–234, August 1986.

[79] B. Wywill and G. Wyvill. Field functions for implicit surfaces. *Visual Computer*, 5:75–82, 1989.

[80] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3D scenes. *Proceedings of SIGGRAPH*, pages 163–170, August 1996.