Avatars and A*Maker for Unicron

Yosep Kim

July, 2005

Project Report to fulfill the Degree of
Master of Science in Computer Science

Abstract

The Department of Computer Science of New Mexico State University (NMSU) is building a collaborative virtual environment named *Unicron* where users from different physical locations can accomplish common tasks in a networked 3D space. The users are represented by a human-like avatar as they interact with one another in the virtual world. This paper discusses an avatar class for Unicron; its design details, implementation, and also an avatar constructor, A*maker.

New Mexico State University
Las Cruces, NM 88003

1. Introduction

Computer technology has greatly advanced in many respects, especially in computer graphics. Three-dimensional graphics, multimedia, and internet advances can be bundled to create a collaborative virtual environment (CVE) that allows people who are physically far from one another to work together. One potential application of CVEs is distance learning. NMSU wants to gives people who do not have access to computer science education due to the physical location, a chance to learn and participate in computer science using a CVE. This is why NMSU's CVE called Unicron was launched [Jeffery05].

An *avatar* is a virtual object representing the participant in the virtual world. The use of avatars is a fundamental part of CVE design [Fraser03]. Users interact with one another using their avatars in Unicron. This report presents the design and implementation of a humanoid avatar for Unicron.

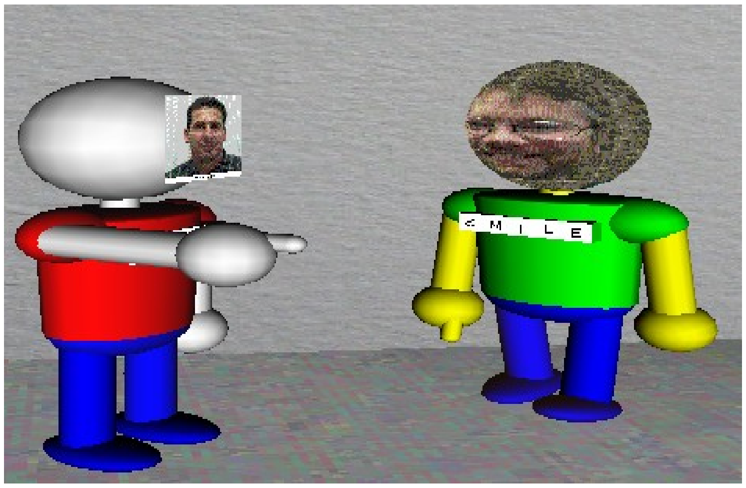


Figure 1. Avatars in action in Unicron

At any given time many users may be logged on to the Unicron server. Each user has a unique avatar which can be distinguished easily from others. Customizing each avatar's appearance according to the user's preference is important. Thus, an avatar constructor A*maker was developed and is discussed in this report. Representing real users is also important. One way to achieve this is to use an actual photo of the users. A*maker helps users build their own avatar using an easy three step graphical user interface (GUI).

This report is organized as follows. Section 2 discusses the design and implementations of the avatar, while the design of A*maker is discussed in Section 3. Section 3 also gives detailed description of an auxiliary GUI program, gif2avt, which helps constructing avatar in conjunction with A*Maker. Section 4 contains the related while Section 5 discusses the future works, and then the conclusion ends the report in Section 6.

2. Avatar

2.1. Design Philosophy

The main design goals of the avatar are simplicity and interactivity. Each avatar consists of a head, a body, two bendable arms and legs, which sums up to some twenty 3D primitives. With this small number of primitives the avatar is simple and yet recognizable as a humanoid. Unicron supports voice chatting and will support video conferencing, which will require substantial amount of network bandwidth; thus it is necessary to minimize the number of network packets required by the avatars and their movements [Jeffery05].

Another aspect of the avatar that requires simplicity is its APIs. From a programmer's perspective, it should be easy for them to create, remove and use the avatar. One example of this simplicity is the method actions(action_name), which takes in a string action_name as its argument; and depending on action_name performs a predefined action. For instance, a call actions("walk_forward") causes the avatar to walk forward. Currently, the avatar can move forward, backward, side to side, turn left or right, raise each arm, sit down on a chair, and grasp or release nearby objects. However, more actions can easily be added with little understanding of the code. This is possible due to the fact that it is designed in an object-oriented fashion, which enables the easy addition of custom-designed actions.

To maximize interaction among users, they must be identified by their avatar. By being able to see whom s/he is interacting, a user can feel more engaged in the 3D environment. Users provide a GIF or JPEG image to represent their face in an ordinary rectangle or texture-mapped within an egg-shaped head, which is more recognizable from the side or rear [Jeffery05]. Since Unicron supports voice chatting, the avatar needs to visually indicate when audio or text chatting is performed [Sharif05]. This feature is discussed in detail in Section 2.3 and 2.4. In order to further support interaction, the avatar can point, for example, instructors can point at examples on the virtual white board for students to follow. The pointing capability is discussed in Section 2.3.

The avatar needs to be created whenever its user logs in to a Unicron server. Physical attributes are stored on the server and replicated at clients. To accommodate this goal each user of the avatar has a physical attribute file with a file extension .avt. For instance, a user with a login johndoe has a physical attribute file, johndoe.avt. The avatar constructor, A*Maker, generates such files for users.

2.2. Class Organization

The class structure is shown in Fig. 1. It consists of seven classes: Avatar, Avatron, Head, Body, Arm, Leg, and avatarParts. The class Avatar specifies abstract methods for required APIs functions for every kind of avatar, including the humanoid avatar described in this report. The main class Avatron conforms to the Avatar class and is responsible for creating and setting up an avatar by invoking the body part classes, Head, Body, Arm, and Leg, and allowing programs to manipulate them using methods in the Avatron. The avatarParts is an auxiliary class that has a set of commonly used methods found in body part classes, thus getting rid of redundancies.

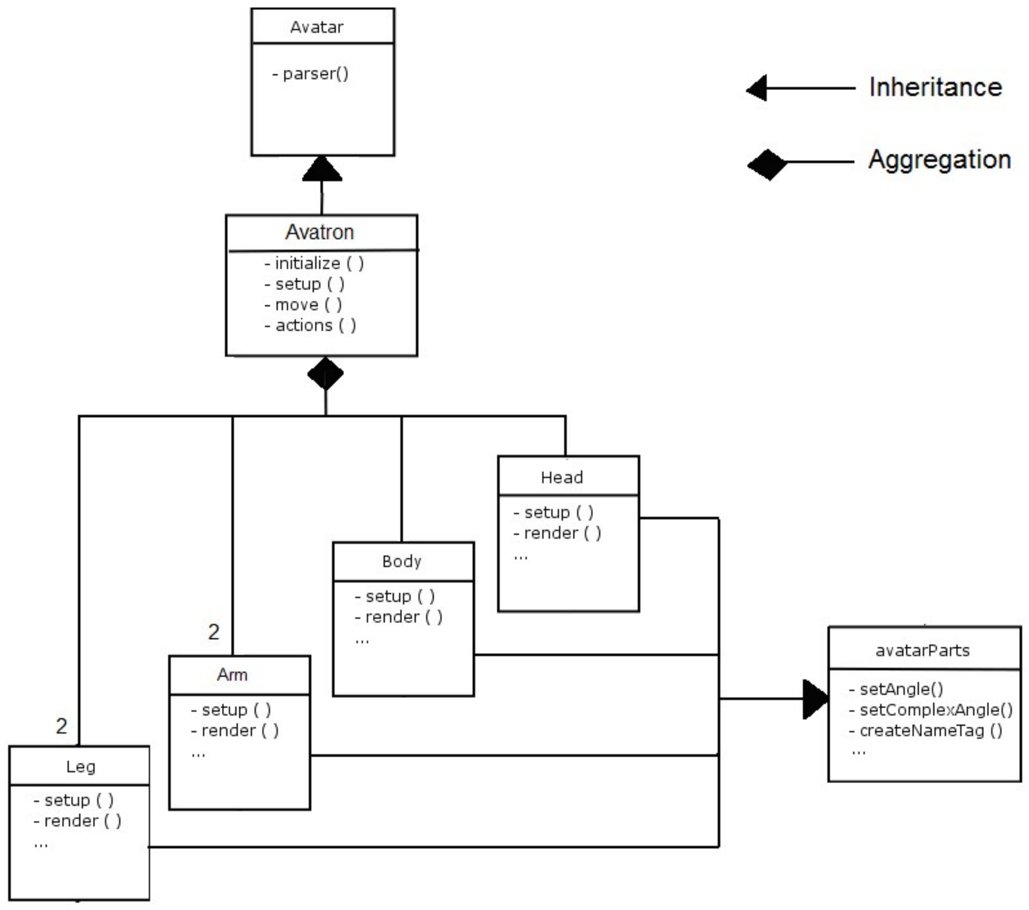


Figure 2. Class Hierarchy Diagram for Avatar

The avatar is composed of six movable body parts: head, body, two arms, and two legs. A female avatar may have chests made up of two additional spheres, which are invoked in the body class. Each of six movable body parts is capsulated in a separate class. For instance, the *head* class is responsible for rendering head and neck and for executing functions such as moving, turning and visually indicating when audio or text chat is performed. A discussion on each class in detail is in Section 2.4.

2.3. Implementation

Unicron and its supportive applications are written in a very high level programming language called Unicon. Unicon is a dynamic typed language that features a 3D graphics facilities based on OpenGL and voice over IP facilities. With these multimedia capabilities and its relatively easy usability that derives from high-levelness make Unicon a language of a choice for developing complex applications such as a CVE. The avatar is developed in Unicon as well. The total number of lines of Unicon code written out for the complete avatar is around 1150. A*Maker, which is also developed in Unicon, is made possible with 1060 lines of code, which includes gif2avt.

Rendering an avatar is done through a series of 3D graphics calls from Unicon 3D library. Such graphics calls are DrawCylinder, DrawSphere, and FillPolygon. When an instance of Avatron is invoked and rendering takes place, an instance of head, body, two arms, and two legs are invoked and rendered in order. All the 3D graphics calls are put into a set, S. From the class Avatron to the class Leg, all 3D calls are put into the set. The following piece of code is excerpted from the class Head, and every 3D call is inserted into the set S.

```
insert(S, PushMatrix())
   insert(S, WAttrib("texmode=on","texture="||face,"texcoord=auto"))
   insert(S, Rotate(270,1,0,0))
   insert(S, Scale(1,1,1.2))
   insert(S, PushMatrix())
      insert(S, Translate(headx,0,0.55))
      insert(S, DrawSphere(0, 0, 0, headr))
```

This is done to accomplish an easy derendering of the avatar. In Unicon every 3D call is pushed into a list called WindowContents from which a 3D scene is constructed. For instance, the WindowContents for Unicon contains all the 3D graphics calls for rendering everything that is in Unicron. Thus, every3D call for the avatar, after rendered in Unicon, is also pushed into the list. To derender the avatar, all it needs to be done is to pop out all the calls in the WindowContents that are in the set S. This causes the avatar to be removed from the Unicron.

When an avatar is rendered, a translation matrix, mv, and a rotation matrix, ro, allows the rendering at the initial location and orientation (i.e. which way is the avatar facing). These two matrices are modified by the move method to reflect the changes in location and orientation of an avatar.

```
method render()
   local S
   S := set()

   insert(S, PushMatrix())
      mv:=Translate(X , origy+2*height , Z )
      insert(S, mv)
      ro:=Rotate(angle,0,1,0)
      insert(S, ro)
      ...
```

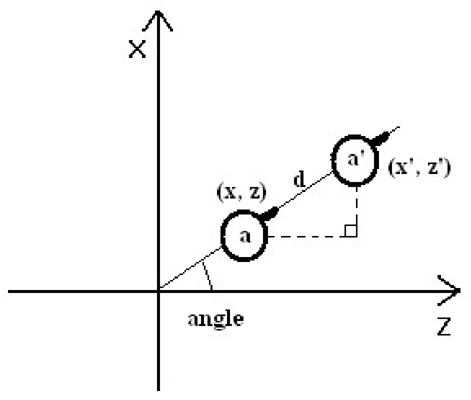
```
method move (posx,posy,posz, angle)
   local degree
   degree := rtod(angle)
   if /ro then fail
   ro.angle:=degree
   mv.x := posx
   mv.y := posy
   mv.z := posz
   ...
```

Movable body parts also make use of the Pushmatrix and PopMatrix pair. Each movable body part, for instance an arm, is capsulated between a Pushmatrix and a Popmatrix. Followed by the Pushmatrix, the initRMatrix method of the superclass AvatarParts class initializes three rotate matrices, one for each axis. The matrices are named romx, romy and romz for x-axis, y-axis, and z-axis respectively. Once this method is called, the graphics drawn are affected by the matrices. These matrices can be manipulated by setAngle(a, d) and setAngleComplex(x,y,z). This topic is discussed in detail in Section 2.8.

The avatar has a set of pre-defined actions such as walking forward and turning left. The following is a piece of code for having the avatar walk forward.

```
origx:=origx+sin(angle)*0.1
origz:=origz+cos(angle)*0.1
move(origx, origy, origz, angle)
if fint < 3 then {
    move_part("right_arm", "fb", 10)
    move_part("left_leg", "fb", 10)
    move_part("left_arm", "fb", -10)
    move_part("right_leg", "fb", -10)
    fint:=fint+1}
else {
    move_part("right_arm", "fb", -10)
    move_part("left_leg", "fb", -10)
    move_part("left_arm", "fb", 10)
     move_part("right_leg", "fb", 10) }
 fint:=fint+1
 if fint = 6 then {
     fint:=0 }
```

First, the new position of the avatar is calculated from the current position using the cosine and sine function. The angle is the orientation of the avatar with respect to the original x-y-z coordinate.



$$x' = x + \sin(\text{angle}) * d$$
$$z' = z + \cos(\text{angle}) * d$$

Then, the position of the avatar is updated by calling the move method with the new position. For making it look as though it is walking, bringing one arm and a leg forward and the others backward are performed.

2.3. Class: Avatar

The class Avatar is an abstract class that defines required functionalities for every kind of avatars. Thus, the class contains abstract methods for such functionalities. This project deals with a humanoid avatar, however, other types of avatars can be introduced. For instance, a class called Penguin may come across.

Figure 3. An instant of Penguin class inheriting Avatar class

Currently, the class Avatar contains two abstract methods: parser and render. The method parser should take a name of an avatar name and reads off the file whose name is composed of name.avt to create the avatar. The method render should draw the avatar in the 3D space. The following is the definition of the class Avatar.

```
class Avatar()
    abstract method parser(name)
    abstract method render()
    abstract method derender()
    abstract method actions(action_name)
    abstract method move(x, y, z, angle)
    abstract method movePartComplex(part_name, x, y, z)
end
```

The detail information on each method is described in the following section, for the following section discusses of methods in the class Avatron, which inherits the Avatar class.

2.4. Class: Avatron

The Avatron is the main class that rounds up the other classes, which makes up an avatar. The following sections discuss API of Avatron in detail.

setup (X, Y, Z)

The setup method instantiates every part of an avatar. Such parts include head, body, left arm, right arm, left leg, and right leg. Chest is instantiated for female avatars. The three numeric parameters are the x, y, z coordinate of the initial location of the avatar.

render ()

Given that all body parts are instantiated this method actually calls the render method of each body part to render the entire avatar. By scaling the avatar using the physical attributes gathered from parsing the property file the appropriate size of the avatar is realized.

derender ()

The derender method pops out every 3D call in the avatar render set S from the WindowContents list for the whole Unicron. This removes the avatar from Unicron.

parser (file)

The parser method reads and parses through a file whose name is passed in as parameter. The parameter is a name of an avatar. Thus, this method adds the ".avt" extension to the name, opens the file, and parses through it to set class variables such as a_name (i.e. name of the avatar), height (i.e. scaling factor along with y-axis for the avatar), color (i.e. skin color of the avatar) and so on.

move (posx, posy, posz, angle)

The move method takes four parameters: posx, posy, posz, and angle. Here, the posx, posy, posz are the new x, y, z location of the avatar needs to be, while the angle is the angle by which the avatar needs to turn. The angle parameter should be in radian.

move_part (name, dir, angle)

The move_part method rotates a particular body part. The parameter, name, is the name of the part that needs to be rotated or moved. The movable parts through this method are right arm, left arm, right leg, and left leg. The second parameter, dir, is the direction in which the part needs to move or rotate, while angle determines how much angle in radian the part needs to be rotated or turned. The possible value of dir is "s" and "fb" for side-to-side and forward-and-backward, respectively. After deciding which part needs to be rotated via name, this method calls setAngle(angle, dir) method of the body part to execute the rotation action according to dir and angle.

move_part_complex (name, angle1, angle2, angle3)

The move_part_complex method accomplishes the same functionality as the move_part, however, instead of using the string value for direction, it uses three angles, one for each x, y, z angle. To use this the subclass must have a method named setAngleComplex().

actions(action_name, optional_parameter)

The actions method takes a string action_name, which is a name of a pre-defined action to be taken and an optional string parameter optional_parameter for such as a name of an object for the part to hold. The following is the table of possible string values of action_name and their corresponding actions.

String Value	**Optional Parameter**	**Corresponding Action**
move_forward	None	Move forward
move_backward	None	Move backward
move_right	None	Move to right
move_left	None	Move to left
turn_left	None	Turn left
turn_right	None	Turn right
raise_right_arm	None	Raising right arm
raise_left_arm	None	Raising left arm
back_default	None	Put every part back to default
toggle_talking	None	Talking on/off
move_pointer_up	None	Move pointer up
move_pointer_down	None	Move pointer down
move_pointer_left	None	Move pointer to left
move_pointer_right	None	Move pointer to right
possess	Name of an Object	Posses Object
unposess	Name of an Object	Unpoess Object
sitdown	Name of an Object	Sit down on Object
standup	None	Stand up

Figure 4. String values and their corresponding action for method actions

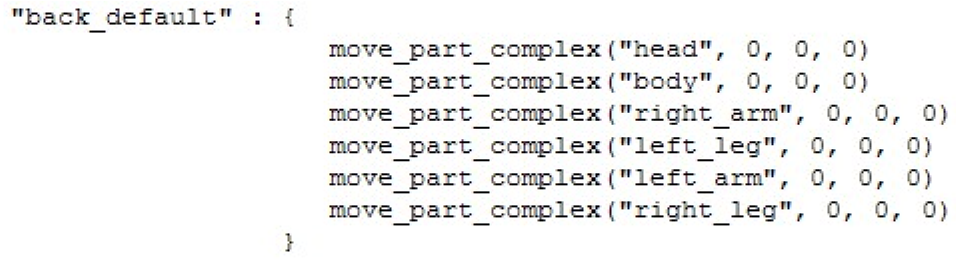
When actions("toggle_talking") is called the first time, it brings forward four red bars around mouth area by calling a method toggleTalking() in the Head class. When it is called again, it puts the bars back. Hence, by repeatedly calling the method an effect of a blinker is accomplished. The detail is given Section 2.5.

Calling actions("possess", Obj) allows the avatar to pick up an object, Obj, on its hand. When Obj is picked up by an avatar, it is put into a list, which contains all the objects being held by the avatar. Only one object can be visible at time, others being kept invisibly. By contrast, calling actions(“unpossess”, Obj) deletes the object, Obj, from the object list, thus, it no longer holds the object.

To have the avatar sit down on an object, Obj, actions(“sitdown”, Obj) is called.. The avatar first bends the knees and then placed on the object. The object being passed must have APIs for returning its desired sitting x-y-z position and the sitting angle for the avatar. For instance, Unicron has a sittable object called Chair, which offers a set of methods such as getX(), getY(), and getZ() for getting x, y, and z value of the position of the object repectively.

Since the avatar can only sit on one object at a time, there is no need of keeping track of multiple sittable object. Calling actions(“standup”) causes the avatar to stand up from an object. When standing up, the knees of the avatar straighten up, and it faces to the direction it was originally facing before it was sat down.

The method utilizes other methods in the class to make the action possible. For example, the following is a piece of code for handling “back_default”.



```
"back_default" : {
                    move_part_complex("head", 0, 0, 0)
                    move_part_complex("body", 0, 0, 0)
                    move_part_complex("right_arm", 0, 0, 0)
                    move_part_complex("left_leg", 0, 0, 0)
                    move_part_complex("left_arm", 0, 0, 0)
                    move_part_complex("right_leg", 0, 0, 0)
                 }
```

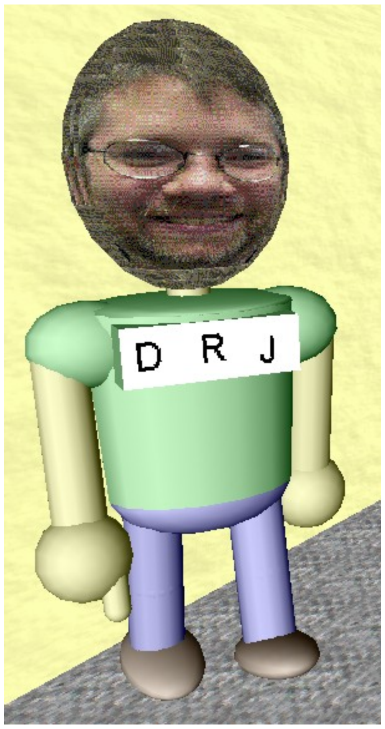
By calling a series of move_part_complex body parts that may have been dislocated are restored to their default position.

2.5. Class: Head

The head class is responsible for rendering the head and the neck of the avatar, as well as the operations that the head performs in order to provide a visual representation of talking.

render (face, color, choice)

The first argument, face, of this method is a name of a GIF file. That represents the user of the avatar. There are two ways to utilize the picture, and it is governed by the third parameter choice. If choice is 2 then the picture will be wrapped around an egg-shaped sphere, otherwise it is put on a square which is sitting on the front side of the head as face by default.



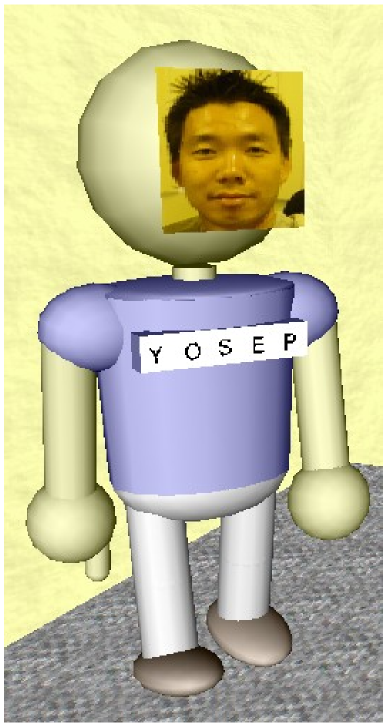


Figure 5. Egghead representation

Figure 6. Blockhead representation

The advantage of the egghead representation is that it looks more realistic than the square one. It also provides more partial information from the side or rear than does the square reprentation. However, modifying a user's picture to accommodate wrapping, as shown below, required extra work.

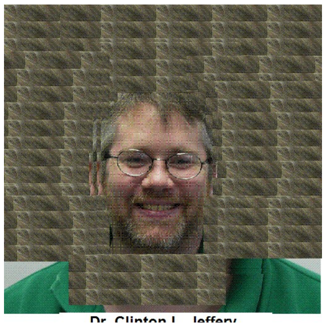


Figure 7. Processed GIF snapshot for Egghead

The second parameter, color, of the method render dictates the color of the head. For instance, the given color of the head of the avatar in <Fig. 8> is light yellow. However, if one wishes to use a texture instead of a color, this is possible. The texture file should be either in GIF or JPEG format.

toggleTalking()

As stated in Section 2.3, the method provides a representation of whether or not the avatar is talking. The following is how such action is done.

When the head class is initialized, a set of red bars is rendered in side of the head sphere. It is not visible at any time until the method toggleTalking is called. Calling toggleTalking moves the cylinders right about where the mouth would be, as shown below.

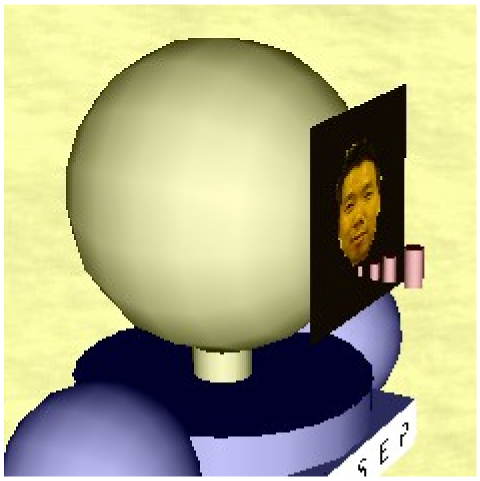


Figure 8.a. TalkingOn

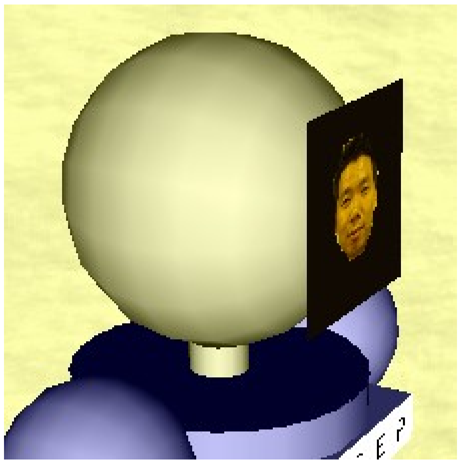


Figure 8.b. TalkingOff

When the toggleTalking is called again, the bars move back to its original position, preventing it from viewing. When the method is called again and again with rapid speed, it looks almost as if it's blinking, creating an effect of talking.

2.6. Class: Body

The *body* class is responsible for rendering the body and the chest of the avatar, respectively. They both have one method in them, which is the *render* method and each render is very similar. Thus, the following section describes the render method for both classes.

render (color)

This method takes one parameter, color, as input. Here, color can be either a color (e.g. white) or a GIF file name. If it is a GIF file name, the file will be used as a texture over the body or the chest.

2.7. Class: Arm and Leg

The class *arm* and *leg* are similar in their design and their operations. Both take a parameter with a value of either “l” or “r”, where “l” is for left while “r” right. When “l”is passed a left arm (or leg) is drawn, while passing “r” gives the avatar a right arm (or leg). When the right arm is drawn, an index finger is also rendered. This is the pointer for the user to point at objects.

Both the class Arm and Leg are responsible for rendering their body parts and managing their orientation at any time. Each of these classes has the render method to render its part and its setAngle method modifies the position and orientation of the part.

render (color1, color2)

The render method takes two colors as parameters. For the arm class first color is skin color, while the second color is for shirt color. As the coloring/texturing scheme for the head and body classes, this one also takes either color or texture. In this method a rotation matrix, ro, is defined. At initial point, it is set to zero, thus there is no rotation. However, the rotation matrix is updated via the following method, setAngle().

2.8. Class: avatarPart

The class avatarPart is a super class for the body part classes described in the previous sections. It contains commonly used methods for the body parts.

initRMatrix()

The initRMatrix method initializes three rotate matrices, one for each axis. The matrices are named romx, romy and romz for x-axis, y-axis, and z-axis respectively. Once this method is called, the graphics drawn are affected by the matrices. These matrices can be manipulated by setAngle(a, d) and setAngleComplex(x,y,z), which are described in the following sections.

setAngle (angle, dir)

Upon taking an angle (in degree) and a direction (either “fb” or “s”) this method updates the rotation matrix defined in the render method to move the body part. As mentioned in section 2.3, the argument dir determines to which direction the body part move, and angle determines by how much it should rotate. At this point the possible values for dir are “fb” and “s”; where the former stands for forward-backward while the latter side-to-side.

setAngleComplex (xangle, yangle, zangle)

The method setAngleComplex accomplishes almost the same thing as setAngle(a, d). However, instead of using a direction this uses values for each angle to be rotated. Thus, this method gives more flexibility in choosing directions, as it can modify the matrices romx, romy, and romz.

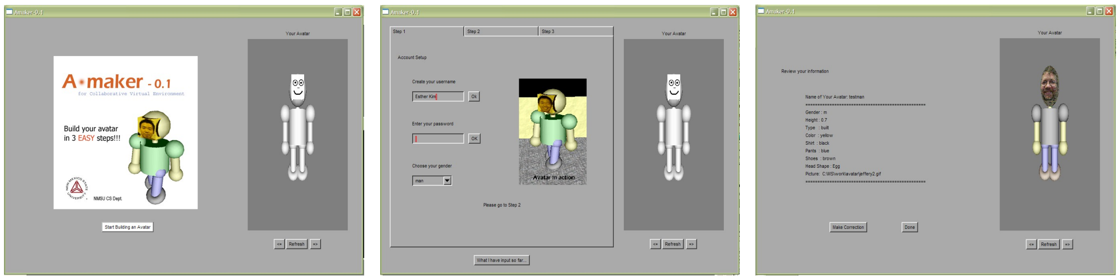
3. A*Maker

An instance of Avatron is constructed from an avatar property file, which specifies the physical body attributes. The avatar property file can be produced by A*Maker, though a user can create or modify it by hand using a text editor.

A*Maker is a GUI application that helps users build their humanoid avatars for Unicron. It offers three easy steps to customize an avatar. This section describes A*Maker.

3.1. Design Philosophy

A*Maker is designed to be simple and user-friendly. A*Maker has three main pages in its GUI, the greeting page, the work page, and the confirmation page. All three pages are shown below.



Esther Kim

man

```
Name of Your Avatar: testman
=====================================================
Gender : m
Height : 0.7
Type   : built
Color  : yellow
Shirt  : black
Pants  : blue
Shoes  : brown
Head Shape : Egg
Picture:  C:\MS\work\avatar\jeffery2.gif
=====================================================
```

Figure 9. Three main pages of A*Maker

A user can easily follow on-page instructions to construct his/her avatar, however, the second page is worth presenting in some detail.

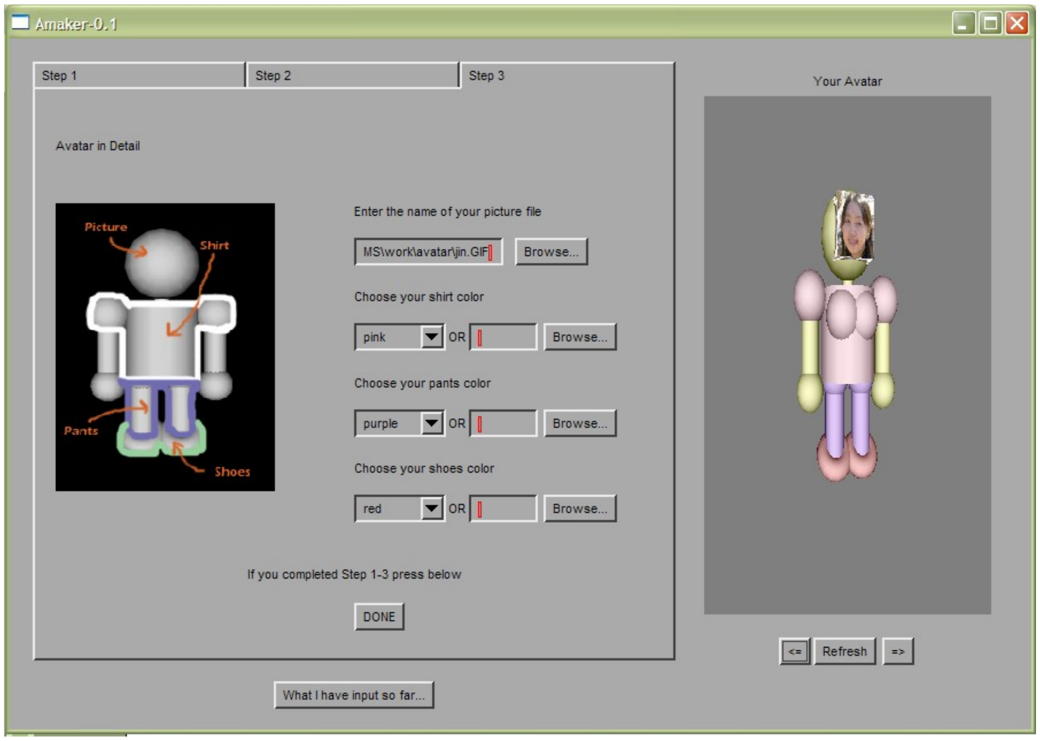


Figure 10. A*Maker in detail

On the second main page a user can follow three easy steps to construct his/her humanoid avatar. As seen in Fig. 12 above, it is divided into two sections. A section on the left allows the user to choose his/her avatar's physical properties, while the right panel shows the avatar being built with the properties. A user can rotate the avatar in progress left or right using arrow buttons below the avatar.

A*Maker writes an avatar property (.avt) file used by the Avatron parser() method.

3.2. gif2avt

gif2avt is a GUI program that lets users create an egghead-compatible snapshot out of a regular snapshot. Aforementioned Avatar lets users choose a type of the avatar's head. It could be either egghead or block-head. For block-head one may use a regular snapshot, however, for egghead one needs a specially edited picture that would be wrapped around the egg-shaped head and still maintain pleasing looks. Producing such an edited picture can be tedious and time-consuming. Thus, an automated procedure of converting a regular snapshot to an egghead-compatible one is sought. This is the motivation behind the development of gif2avt.

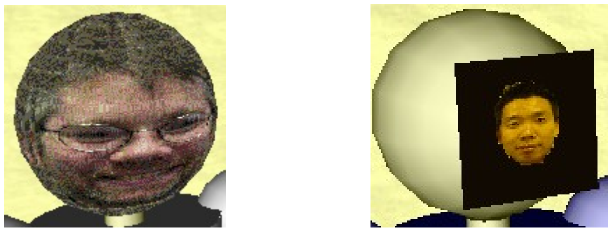


Figure 11. Egghead Vs. Blockhead

gif2avt is a prototype but enough to achieve the goal of proving a regular face shot to egghead-compatible conversion.

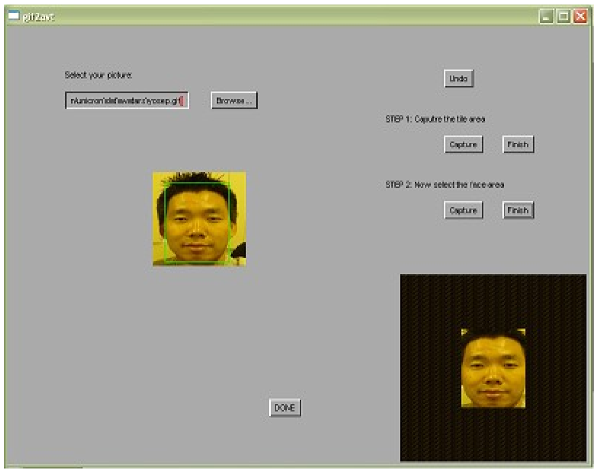


Figure 12. Snapshot of gif2avt

A snapshot selected by the user is placed on the center of gif2avt. The snapshot is resized to 128 pixels by 128 pixels. Then the user can select a rectangular area (a tile) from the picture to be the background of the egghead-compatible faceshot. Once the user selects the tile, it is copied over and over on the bottom right corner of gif2avt in the size of 256 pixels by 256 pixels. Then the user can select the face from the picture, which would show up on the center of the tiled area on the bottom right corner.

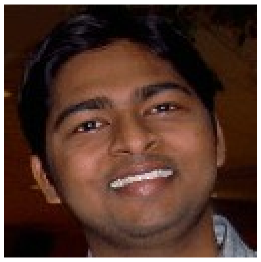


Figure 13. Before the conversion

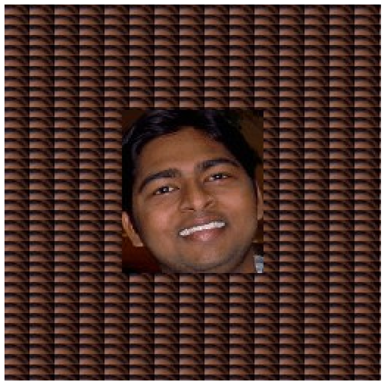


Figure 14. After the conversion

The following section discusses in detail an avatar attribute property file.

3.3. Avatar Attribute Property File

In order to recreate avatars at different logins it is important to store all the attributes of the avatars so they can be referenced during process of recreation. The format of an avatar attribute property file is simple. A field is followed by appropriate information, as shown below.

```
NAME=yosep
```

Comments are allowed in the file, and must start at a new line. Comments should start with character #. Comments that start with # followed by @ are the system comments. That is, they were written by A*Maker to help users to understand the contents of the file. In an avatar attribute property file the order of fields must be observed. The following is the order of the fields with example information.

```
NAME=yosep
GENDER=m
HEIGHT=0.8
#@ This avatar's body type is Average
XSIZE=0.7
YSIZE=0.7
ZSIZE=0.7
SKIN COLOR=yellow
SHIRT COLOR=white
PANTS COLOR=blue
SHOES COLOR=black
#@ This avatar's head shape is Block
HEAD SHAPE=1
```

FACE PICTURE=yosep.gif

The fourth and the twelfth line from top starting with #@ are the system comments, which each describing what subsequent lines are.

NAME is the name of the avatar, which is the first half of the file name, as the avatar attribute property file name is [avatarname].avt. GENDER can be either “m” for man or “w” for woman. HEIGHT is the scaling factor along y-axis, and the following categorization should be used. Here, Height is a desirable height for the avatar, and HEIGHT is the actual value that should be applied in the avatar property file. Currently, HEIGHT is being asked in both centimeter and inches. When inches are used the value is converted to a centimeter equivalent. The input value being in centimeter is multiplied by 0.004. For instance, for the input value 180 cm (or about 71 inches) HEIGHT comes out to 0.72, which is suitable for avatars in Unicron scaling.

One can modify HEIGHT with a value other than ones mentioned in the categorization above. For instance, if one wishes her/his avatar to be extra small, one can set HEIGHT=0.3.

XSIZE, YSIZE, and ZSIZE are a scaling factor along x, y, and z axis, respectively, on the avatar. These values are determined by the categorization used by A*Maker. However, YSIZE is currently the same as HEIGHT. There are six categories in the categorization, which are “very slim,” “moderately slim,” “average,” “built,” “big,” and “large.” The following is the categorization.

```
For very slim,
        XSIZE=0.6
        ZSIZE=0.7
For moderately slim,
        XSIZE=0.65
        ZSIZE=0.7
For average,
        XSIZE=0.7
        ZSIZE=0.7
For built,
        XSIZE=0.75
        ZSIZE=0.7
For big,
        XSIZE=0.8
        ZSIZE=0.7
For large,
        XSIZE=0.9
        ZSIZE=0.8
```

Again, these values can be changed by users at will.

SKIN COLOR, SHIRT COLOR, PANTS COLOR and SHOES COLOR can either be a name of a natural color such as white, black, or a path of a GIF file. A list of colors that one can choose from is attached as an Appendix.

HEAD SHAPE, as discussed in Section 4.4.1, can either be an integer 1 or 2, as 1 is for block shape, while 2 for egg shape.

FACE PICTURE should be a path of a GIF file that would be placed on the head to represent who the user is.

This section discussed A*Maker and the avatar attribute property file. The following section describes a small program that runs off A*Maker, which is responsible for creating a egghead compatible picture from a regular snapshot.

5. Related Work

There have been many CVEs created recently for different purposes, such as recreational, industrial training and education. Each of the CVEs has avatars for users to use to navigate through the 3D space interact with one another. The avatars from some of the CVEs follow a specification, and most widely used specification is H-Anim.

H-Anim is a specification of an abstract representation for modeling three dimensional human figures. The main website of H-Anim states that the International Standard describes a standard way of representing humanoids that, when followed, will allow human figures created with modeling tools from one vendor to be animated using motion capture data and animation tools from another vendor [H-Anim04]. H-Anim makes use of virtual reality modeling language (VRML). H-Anim is widely used for developing humanoids, even in CVEs. Using the H-Anim specification a group of researchers were able to develop a relatively complex face model for an avatar that displays emotions through facial expressions [Fabri02]. However, it is hard to represent different users using this model. Most video games use complex avatars, which require substantial amount of effort and work to design and create one. Therefore, such games offer only a few different avatar models with different identities. Unicron, however, needs to represent each user uniquely due to its nature. A research that involves real-time video streaming by capturing a user's face via camera does solve such problem [Capin98]; however, it is expensive in cost, for it requires having extra peripheral devices including cameras and more video and networking capacities. Thus, the current design serves its purpose as far as simplicity and interactivity concern.

6. Future Work

Studies on CVEs repeatedly suggest that nonverbal communication such as body gesture is very important in interaction between the avatars thus the users [Vuilleme00]. However, at current level the avatars for Unicron cannot support such communication mechanism, because they do not have facial muscles or elbows. This also brings up another shortcoming of the avatar. Due to the simplicity the avatars are not realistic in appearances. This can be fixed by adding more polygons to make avatars look more realistic. However, this brings up a problem in network traffic, for more polygons might cause more packets being sent for avatar position updates. This leaves a research opportunity to develop a better, more efficient way to synchronize the positions of the avatars in Unicron. If this network issue is resolved, the avatar can get complex to look more realistic, and to support different body gestures.

Another opportunity for future work is to better represent the avatar when its user is talking. Currently, the blinker serves its purpose; however, more visual and realistic way is desired.

There is a lot of room for improvement for gif2avt. The current design is very crude and primitive. It uses hardcoded values during the conversion process. Thus it is important to look for ways to dynamically set the size of the converted picture that result in the most recognizable and appealing egghead shot. In order to achieve this, an optimal size of the face, relative to the size of the background is needed. Inappropriate sizes of the face result in excessive stretching, which makes the egghead representation look unpleasant.

A better way to capture the face and the tile area is being renovated. The current way makes use of a simple rectangular capturing scheme. However, the rectangular picture looks out of proportion, when stretched over the egghead. Thus, capturing the face or the tile by connecting a series of points selected is being researched.

6. Conclusion

This report proposed and discussed the specification of avatars in Unicron. The design and the implementation of the prototype humanoid avatar and its constructor A*Maker was presented in detail. The avatar developed for this project is simple and represents users well, but it lacks realism. Communication mechanisms allowing better nonverbal interaction should be developed for future avatars in Unicron. More 3D graphical detail should be done to make avatars more realistic so that users feel more attached to their avatars. Despite the shortcomings, the avatar presented here serves its purposes well.

Bibliography

[Jeffery05] Jeffery, Clinton; Mohamed, Shamim; Pereda, Ray; and Parlett, Robeter. Programming with Unicon. Draft manuscript from http://unicon.sourcefourge.net

[Winkler04] Winkler, Wynn. Creating Collaborative Virtual Environments Using Unicon. NMSU CS Department Technical Report #######

[Martinez04] Martinez, Naomi; and Jeffery, Clinton. Unicon 3D Graphics User's Guide and Reference Manual. Unicon Technical Report #9a

[Fraser03] Fraser, Mike; Hindmarsh, Jon; Benford, Steve; and Heath, Christian. Getting the picture: Enhancing avatar representations in collaborative virtual environments. Inhabited Information Spaces, pp.133-150, London: Springer-Verlag, 2003.

[Fabri02] Fabri, M; Moore, D.J; Hobbs, D.J. Expressive Agents: Non-verbal Communication in Collaborative Virtual Environments, in Proceedings of Autonomous Agents and Multi-Agent Systems 2002 (Embodied Conversational Agents Workshop), July 2002, Bologna, Italy

[Sharif05] Sharif, Ziad. High Level VOIP API for Unicon Language. Unicon Technical Report #10a

[Capin98] Tolga K. Capin; Igor Sunday Pandzic et al. Realistic Avatars and Autonomous Virtual Humans in VLNET Networked Virtual Environments., in Virtual Worlds on the Internet. Earnshaw R, Vince J eds. IEEE Computer Science Press. 1999

[H-Aim] ISO/IEC FDIS 19775-1:200x, *Information technology — Computer graphics and image processing — Extensible 3D (X3D) — Part 1: Architecture and base components.*