



**159.794 159.795**

**3D modeling of Human Features**

**(3D Hand Model by Povray)**

**Superior: Andre L. C. Barczak**

**Student: Felix (Feng) Xie**

## **Abstract**

To achieve the goal of creating a 3D hand model for hand tracking system, this paper introduce some classic 3D hand models and these models are use for hand tracking system, also introduce American Sign Language (ASL). It also demonstrates three 3D hand models simulating natural articulate hierarchical of human hand. These models are using ray-tracing language Povray as a 3D modeler. These models can simulate the gesture in American Sign Language (ASL). And use by a C++ program, in order to create a serial still image for the gesture in ASL. All of these models are skeletal models, although, they can achieve high level of virtual realism. For feature work, models' realistic will significant enhance by applying skin texture.

## Table of Content

<b>ABSTRACT.....</b>	<b>1</b>
<b>TABLE OF CONTENT .....</b>	<b>1</b>
<b>1.0 INTRODUCTION .....</b>	<b>3</b>
1.1 Hand Tracing .....	3
1.2 Povray Introduction .....	6
<b>2.0 LITERATURE REVIEW .....</b>	<b>9</b>
2.1 Computer Graphics .....	9
2.2 Medicine.....	11
2.3 Computer Vision (Machine Vision) .....	12
2.3.1 Classic Kinematical Model .....	12
2.4 Other Models Basic on 27 DOFs Model.....	15
2.4.1 12 DOFs Hand Models .....	16
2.4.2 8 DOFs Hand Models .....	16
2.4.3 20 DOFs Hand Models .....	17
2.4.4 Hand models Create by Meatball.....	18
2.5 American Sign Language (ASL).....	18
<b>3.0 METHODOLOGY .....</b>	<b>20</b>
3.1 Povray.....	20
3.1.1 Camera and light .....	21
3.1.2 Cylinder: .....	22
3.1.3 Cone.....	23
3.1.4 Plane .....	24
3.2 Basic Transformation .....	24
3.2.1 Translation: .....	25
3.3.2 Rotation .....	25
3.3.3 Scaling .....	26
3.3 Advanced features.....	27
3.3.1 “Declare” syntax.....	27

3.3.2 “Merge” syntax.....	28
3.3.3 Intersection .....	28
3.3.4 Quadric.....	29
<b>3.4 Super quadric.....</b>	<b>30</b>
3.4.1 Ellipsoid.....	30
3.4.2 Cone.....	31
3.4.3 Cylinders .....	32
3.4.4 Pair of planes .....	32
3.3 Implement quadric in Povray.....	33
<b>4.0 MODEL IMPLEMENT .....</b>	<b>36</b>
4.1 Hand Model 1 (Sphere & Cone).....	36
4.2 Hand model 2 (Quadric Ellipsoid) .....	37
4.3 Hand model 3 (Quadric Ellipsoid & Cone) .....	38
4.4 Hierarchical Articulate of Three Models.....	40
4.5 Create Gesture Image by Models .....	42
<b>5.0 RESULT .....</b>	<b>43</b>
5.1 Model one (Sphere & Cone).....	43
5.2 Model Two (Quadric Ellipsoid) .....	44
5.3 Model Three (Quadric Ellipsoid & Cone) .....	45
5.4 Feature Work.....	45
5.5 Summary .....	46
<b>5.0 CONCLUSIONS .....</b>	<b>48</b>
<b>REFERENCES .....</b>	<b>49</b>
<b>Appendix A</b>	<b>Model Template One Source code</b>
<b>Appendix B</b>	<b>Model Template Two Source code</b>
<b>Appendix C</b>	<b>Model Template Three Source code</b>
<b>Appendix D</b>	<b>C++ Program Source code</b>
<b>Appendix E</b>	<b>asl.txt File (Gesture Information)</b>

## **1.0 Introduction**

As the computer technology developed, ubiquitous human-computer interaction (HCI) has become important for computer technology reaching. The difference between the ubiquitous human-computer interaction and the human-computer interaction we are using today is that, it abandons keyboard and mouse. Instead it uses some hand posture, such as sign language, body language, voice, and so on. In this case, the technology of gesture recognition or voice recognition by computer is very important. They could help the user communicated with the computer.

Using camera or digital eye to capture hand/body gesture to communicate with computer is one of most popular research topic. And a lot of researches about this topic are using 3D-base gesture tracking technology. This technology requires a 3D model to help computer understand the captured image.

### **1.1 Hand Tracing**

To capture human hand motion or hand posture analysis are very interesting and rapid development research area in recent years. There are two approaches to capture hand motion, one is glove-based techniques and vision-based techniques.(Guan, Chua, & Ho, 2001) Although mechanical glove (glove-based) can capture hand motion in real time, but they are very expensive and need special equipment to capture the movement. Meanwhile, the equipment will inhibit finger or hand's movement. The

vision-based techniques can solve these problems, any equipment can capture the images or animation for hand's movement, can be considered as the input source. Usually it is a camera or digital eye. The devices for image capture are cheap and these equipments will not inhibit hand's movement. The only one problem for vision-based techniques is what kind of method use for the movement capture. Vision-based techniques can be classified into two types: appearance-based approach and 3D model-based approach.(Vladimir I. Pavlovic, July 1997)

In computer vision aspect, 3D model-based approach knows as well as analysis-by-synthesis approach. Most of time 3D human hand model is used for analysis-by-synthesis tracking and recognition of the hand motion or gesture(C.Wren, A.Azarbayejani, T.Darrell, & A.Pentland, 1996; R.Koch, 1993). The main idea about the analysis-by-synthesis approach is to analyze the body's posture by synthesizing the 3D model of the human body in question and then varying its parameters until the model and the real human body appear as the same visual image.(Vladimir I. Pavlovic, July 1997)

Obviously, a good 3D hand model can improve the speed and accuracy of hand tracking analysis. The more we know about real human hands, the better 3D hand model we can create. As a human being, after thousand years of evolution, human hands are very complex and delicate. And also they are highly articulate. First, each finger consists of three bones and three joints, all fingers joint connect with palm.

Each finger joint has its own motion. Second, all finger bones joint together by finger joints, one finger joint's motion could cause serial finger activity, which is complex kinematics and articulate activity. Finally, human hands are also highly constrained; there are some limitations for finger joints. These characteristics of human hands make them very hard to model.

There are lots of 3D hand model for computer graphic, computer animation and medicine research, why we have to create another 3D hand model for 3D model-based approach? Most of time, 3D hand model used in computer graphic, animation or medicine has complex surfaces; they are created by some technologies, such as NURBS or nonuniform rational b-splines. Complex surface can make the hand model looks very real. (N.Magenat-Thalmann & D.Thalmann, 1990). But it will cause problem for the analysis-by-synthesis approach. The problem is hard to render the model in real-time, when 3D model is too complex. The more complex model, the more delay it will cause, when the program running. On the other hand, when the 3D hand model creates by NURBS or Constructive solid geometry (CSG) or Polygon for the computer graphic or animation, there is big chance the 3D hand model will have a lot of noises in the image, just as, the curve edge, roughness surface. Furthermore, if the realistic 3D hand model is used at real-time hand posture tracing, then for any gesture change, the program need to handle huge amount of parameters data. All of these will reduce the efficiency of the analysis-by-synthesis approach. So, different kind of 3D hand model from computer graphic for analysis-by-synthesis approach is

necessary.

## **1.2 Povray Introduction**

Povray is a good quality; freely open source ray-tracing software, which use for scene description in 3D computer graphic, other similar language, such as Lightwave Modeler, Rhinoceros 3D, and Moray. It is available in PC, Macintosh and UNIX computer platforms. It is perhaps one of the most commonly used ray-tracing software today, because of its relative ease of use, cheap price and the high quality.

Before, we introduce more about the Povray; we need to explain what ray-tracing software in 3D computer modeling is. Ray-tracing is a method of creating visual art and a rendering technique that calculates an image of a scene by simulating the way ray of light traveling the real world (The\_Online\_POV-Ray\_Tutorial\_ThinkQuest\_Team, 1996). In the real world, if we see some scenes, some basic objects should include in the scenes, a light source, an object or model, and a viewer (cameras). When the rays of light, which is a semi-infinite line, emitted from light source, some of these rays of light go directly from the source through the camera lens or viewer's eyes, and some of the rays strike and illuminate the object. These rays of light may reflect off of the objects and hit the camera lens or view's eyes. That is the usual way that we can see things. Ray tracing software is using calculation of computer to simulate this physical effect process, but it does its job backwards.

To start the ray tracing software, user need to locate the camera or viewer's position, where to put the light source and the object, which will be seen by the camera, and also need to specifies the surface texture properties of objects, their interiors (if transparent) and any atmospheric media such as fog, haze, or fire. As mention before, the ray tracing software do its job backwards from the real world, which means it will start from the pixel will appear in the final image.

For each pixel in the final image or will be seen by the viewer, a ray of light will be shot from this pixel into the scenes, if one of these "viewing rays" hit the object or model in the scenes, then this point of model will be represented in the final image. Next, the computer will start to calculate this point's color and create a ray sent backward to the light source, in order to determine the amount of light coming from the source. And it also can determine whether the surface point lies in shadow or not.

The method above explains the technology using by Povray application. The reason the software trace the light ray backwards, instead of starting at the light source, is for efficiency's sake. In the real physical process, the light ray will not end up on camera or viewer's eyes. If the computer calculates the light ray like the real light process, from the light source, hit the object in the scenes, and goes though the camera. There is a big problem, as a light source, it can create infinite light ray in logical and there is no way to know which light ray is useful for final image until this light ray hit the camera. If computer does its job like this way, it will take a very long time to get the

result for all pixels in final image. The calculation for light rays, which will not display in final image, will waste too much computer resource and time. By tracing the light rays backwards, beginning at the computer screen, the software can assure that every light ray it calculates is one you care about, because it knows that it will end up on your screen. Although, the computer is very power today, but rays of light is infinite. The computer just can calculate the ray-tracing model closely approximation to the physical world.(Edward, 1996)

### **1.3 Overview**

This paper will create a 3D hand model using ray-tracing language Povray. It is a 21 Degrees of Freedom (DOFs) articulate kinematics 3D hand model. And this model can simulate the hand gesture for American Sign Language (ASL), in order to train the hand tracking system or use directly by the hand tracking system. The model created in this paper is rendering under Povray 3.6 vision for Linux platform.

In section 2, it will present a brief literature survey of 3D hand model, analysis the structure of hand model. Some basic technology and methodology will be introduced in section 3. Section 4 will be the implantation of 3D hand model base on methodology description in previous section. Section 5 discusses the advantage and disadvantage for each of three 3D hand models. The conclusion are presented in section 6.

## **2.0 Literature review**

There are a lot of studies had been done about 3D human hand models for different proposes. Therefore, modeling of human hand can be classified in to three disciplines: computer graphics, medicine and computer vision (J.J.Kuch & T.S.Huang, 1994a). This section will descript the useful of 3D model in computer graphics. After that, it represents some material on medicine about 3D model. Finally, as the most important in part in 3D hand model creation, it introduce several articulate 3D hand models.

### **2.1 Computer Graphics**

In computer graphics aspect, the most important require for the 3D human hand model is as realistic as possible. That is the primary goal for the computer graphics and computer animation modeling. To create a high quality realistic 3D human hand model, the first job will be studying the detail of human hand.

The human hand consists of a broad palm (metacarpus) with five digits, attached to the forearm by a joint called the wrist (carpus). Four fingers on the hand are located at the outermost edge of the palm. These four digits can be folded over the palm, this allows for the holding of objects, and furthermore the grasping of small objects. (Wikipedia.com, 2006a) These four fingers are index finger, middle finger, ring finger and little finger or pinky. The thumb (connected to the trapezium) is located on one of the sides, parallel to the arm. The thumb can be easily rotated 90°, on a perpendicular

level compared to the palm, unlike the other fingers which can only be rotated approximately 45°. (Wikipedia.com, 2006a)

3D hand model for computer graphic and computer animation usually creates by thousands of smooth polygons, skin deformations at the joints, and the model will be cover by life-like skin texture, in order to represent a real human hand. (E.Catmull, 1972; H.Rijkema & M.Girard, 1991; J.Gourret, Thalmann, & D.Thalmann, July 1989; N.Badler & M.Morris, 1982; N.Magenat-Thalmann, R.Laperriere, & D.Thalmann, 1988). Now, some successful human hand model incorporate some human's features, such as nails and skin bulging. These models can represent the detail on human hand. Due to the realistic detail, people can easily to know which model is female hand model, which is male hand model, and the age of the hand.

Usually these kinds of 3D hand models are created by professional 3D object software. Such as, AutoCAD, 3D Studies MAX, Poser and so on. In old days, the 3D hand data using in computer graphic and computer animation can get from electromagnetic sensor, such as the Polhemus 3SPACE (J.Floey, Dam, S.Feiner, & J.Hughes, 1990). In order to acquiring the accurate 3D hand model data, usually a plaster caste of human hand will be requiring. Because of the long time scan process and hand model need to be still in the same pose for minutes. It is hard for hand poser, even for the professional poser.

Commonly, realistic 3D human hand models are using in computer graphic and computer animation aspects. But now it also has been used in medical disciplines and computer vision applications.

## **2.2 Medicine**

3D human hand models are very useful for the medical research. In medicine aspect, researchers will not only concern about the out side of the model, they also concern about the inside of the model, the structure of the model. For medical research, it needs an accurate bone and tendon, correct joint articulation model. So, most of these 3D hand model's data are get from X-ray or Computed tomography (CT) (W.Cooney, M.Lucca, E.Chao, & Inscheid, December 1981), (D.Thompson, W.Buford, L.Myers, D.Giurintano, & J.Brewer, August 1988). Or cadavers (W.Buford, L.Myers, & A.Hollister, November 1989; W.Cooney, M.Lucca, E.Chao, & Inscheid, December 1981). Most of these 3D human hand models will be use in clinical and medical education, such as tendon displacement and range of motion (ROM) of joints (D.Thompson, 1981; J.J.Kuch & T.S.Huang, 1994b). In 2003, some researches present a human hand model with underlying anatomical structure. (Albrecht, Haber, & Seidel, 2003) this 3D human model's movement is controlled by the muscle contraction values. They employ a physically based hybrid muscle model to convert these contraction values into movement of skin and bones.(Albrecht, Haber, & Seidel, 2003) As a result, this model can simulate the muscles deform of human hand, and most of the hand motion.

## **2.3 Computer Vision (Machine Vision)**

As mention in introduction, the main idea about the analysis-by-synthesis approach is to analyze the body's posture or hand posture by synthesizing the 3D model of the human body in question and then varying its parameters until the model and the real human body appear as the same visual image.(Vladimir I. Pavlovic, July 1997). Thereby a good 3d hand model with less parameters and realistic simulation can increase the tracing algorithm's accurate and speed. There is a most populate 3D hand model using in hand posture tracing.

### **2.3.1 Classic Kinematical Model**

This hand model is created by Jintae Lee and Tosiyasu L. Kunii in 1995(Jintae & Kunii, 1995). This is 27 degree of freedoms (DOFs) hand model. What is DOFs? Human hand is a complex articulated structure; include comprising bones, connecting bones, muscles, tendon, and covering of skin. Human hand content 5 fingers, each finger consists of three joints. For each finger, it can do some movements, such as flexion, twist, directive or spherical. After analysis, these movements can be considering as the movement of each finger joints. Then, these movements finally can be concluding into flexions/ extensions and adduction/abduction for each joint. Degree of freedom (DOF) is the activity ability of each joint. So that, for each joint it will have one degree of freedom, which is flexions/ extensions. Or it will have two degrees of freedoms (DOFs), which are flexions/ extensions and adduction/abduction.

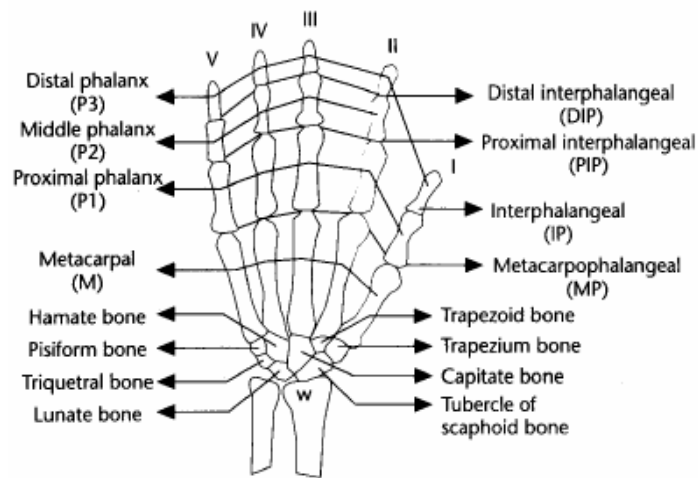


Figure 2.3.1.1: Human hand skeleton. (Jintae & Kunii, 1995)

As the figure show human hand skeleton, it consists of 27 bones. They can be dividing into three groups:

Carpals (wrist bones-eight bones)

Metacarpals (palm bones-five bones)

Phalanges (finger bones-fourteen bones)

In order to create a realistic human hand model analysis the constraints of human hand is essential. Most of the joints, which connecting to carpals have limited degree of freedom (DOF). But the others are different, for example, except for the thumb; there are 2 degree of freedom (DOF) for metacarpophalangeal (MCP) joint, and 1 degree of freedom (DOF) for proximal interphalangeal (PIP) joints and distal interphalangeal (DIP) joints. For the thumb, there are 2 degree of freedom (DOF) for trapeziometacarpal (TM) and metacarpophalangeal (MCP) joint, and 1 degree of

freedom (DOF) for the interphalangeal(IP) joint. As a result, there is 21 degrees of freedom (DOF) for 5 fingers and plus 6 degrees of freedom (DOF) to position and orient it. There are 27 degrees of freedom (DOF) for human hand.

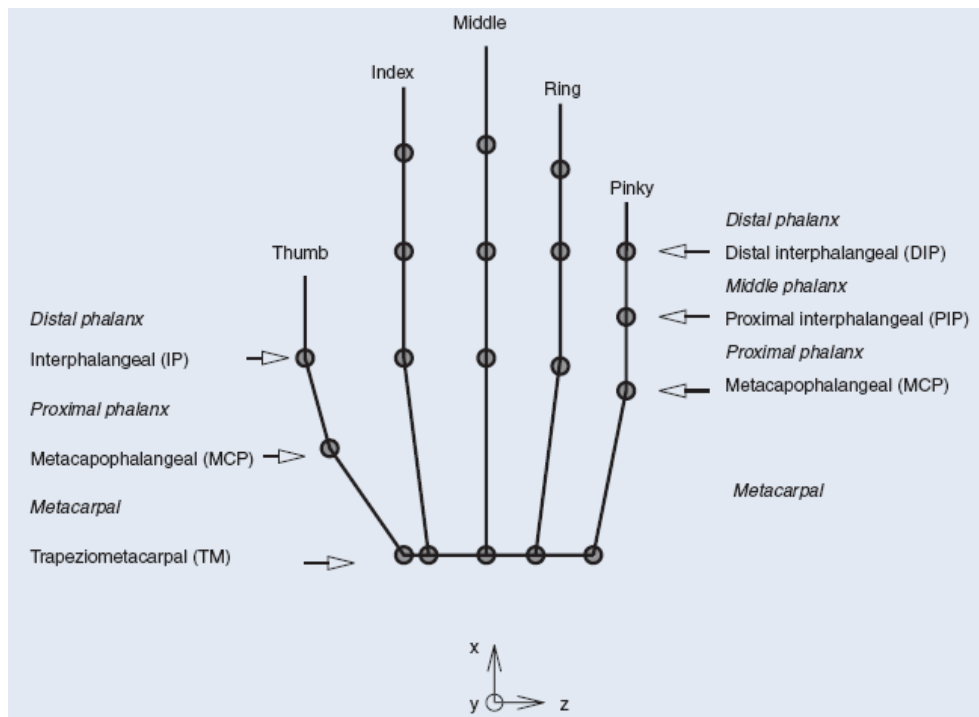


Figure 2.3.1.2: Skeleton-based model of the human hand (Pavlovic, Sharma, & Huang, 1997)

The other important thing to know about is the constraints on joint movements, which is the notion of dependability between the movements in neighboring joints. There are 5 constraints for human hand. First, because the MCP of four fingers (except thumb) can flexions/ extensions and adduction/abduction, and the PIP and DIP joints of these four fingers only can flexions/ extensions in the same direction. In that case, all bones of this finger will be in the same plane. Second, the joint angles of the DIP and PIP joints have a dependency represented by  $Q_{dip} = (2/3)Q_{pip}$ . Third, the joint angle limits

of the MCP joints depend on those of the neighboring fingers (Figure: 2.3.1.3). Fourth, the MCP joint of middle displays limited adduction and abduction.

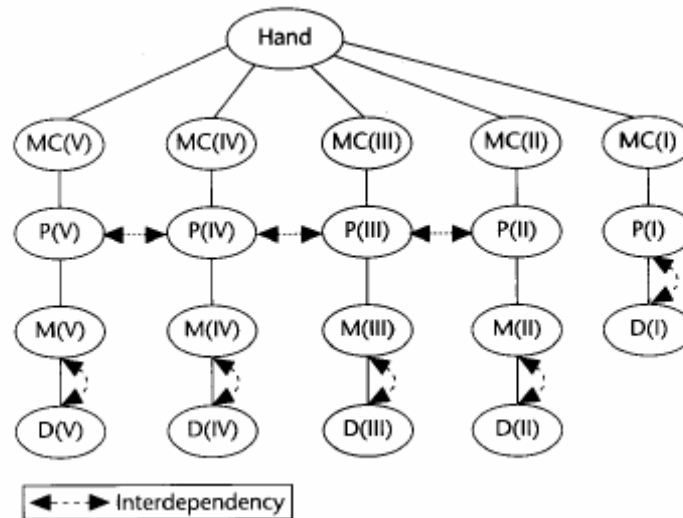


Figure: 2.3.1.3 Hand tree.

Each node represents a segment of the hand. Inter dependency in movement between segments is represented by the dashed lines drawn between the corresponding nodes. There are about 30 degrees of freedom for human hand and changes shape in various ways by its joint movements. (Jintae & Kunii, 1995)

After this hand model, the structure of human hand should be clear now. But this model is too complex for some simple cases, and other researchers have done some modifications for this model.

## 2.4 Other Models Basic on 27 DOFs Model

27 degree of freedom (DOF) model is a base for the analysis-by-synthesis approach. This model helps researchers to create their own model for analysis-by-synthesis approach. Some time for special use, the researcher doesn't need that much Degree of Freedom (DOFs), they reduce some DOFs, in order to achieve higher performance for

the hand tracking system.

#### **2.4.1 12 DOFs Hand Models**

The researchers from Nanyang Technological University in Singapore, they developed a 12 DOFs 3D hand model base on 27 DOFs model. They analyze some constraints of the 27 DOFs hand model and reduce to 12 DOFs model without any significant degradation of performance. (Guan, Chua, & Ho, 2001)

In their model, they use the constraints from RiJpKema(H.Rijpkema & M.Girard, 1991) and Kuch and Huang (J.J.Kuch & T.S.Huang, 1994a), successfully omit 3 DOFs for each finger(Except thumb), and omit 3 DOFs for the thumb. That means there are only one DOF for four fingers, and two DOFs for the thumb. That is 6 DOFs for all five fingers, plus 6 DOFs for the translational and rotation of the wrist. Then, there are only 12 DOFs in their hand models. According to their report, this model didn't significant degradation of the performance.

#### **2.4.2 8 DOFs Hand Models**

Here is the other model from Coventry University in UK; they use this model for breast self-examination (BSE). Breast Self-Examination (BSE) is a non-invasive, self-administered and simple screening procedure for detecting breast cancer at early stage. They simplified the 27 DOFs 3D hand model to 8 DOFs 3D hand model. And this model is especially adapted for use with the breast self-examination system.

The 8 degree of freedom (DOF) is simplified from 27 DOFs model too. According to the techniques of breast self-examination (BSE), the model does not need to use little finger and thumb, the degree of freedom for these two fingers are equally ignored. Meanwhile, the index finger, middle finger and ring finger will joint together and treated as one finger f. The degree of freedom (DOF) for distalinterphalange(DIP) and proximal inter phalange (PIP) from these three fingers will be ignored. Then there only 2 degree of freedom (DOF) left in meta carpophalangeal(MCP) for finger f. and plus 6 degree of freedom (DOFs) for the wrist( 3 for translation for the hand in the x, y and z directions and 3 for wrist rotation), there are 8 degree of freedoms (DOFs) for the whole hand.

### **2.4.3 20 DOFs Hand Models**

The researchers in university of southern California describe 20 degrees of freedom (DOFs) 3D hand model. This model is use for human hand motion detection; it is also base on 27 degrees of freedom (DOFs) hand model.

In their model, it consists of 15 joints. For each finger, it consists of 3 joints and 3 links. For the joint connecting palm and finger, it has 2 degrees of freedom and other joints have single degree of freedom (DOF). Then, totally this model has  $(2+1+1)*5=20$  degree of freedom (DOF).

#### **2.4.4 Hand models Create by Meatball**

A realistic virtual hand modeling using super quadric is creating by state key lab of CAD & CG in ZheJiang University, China. They create a sophisticated virtual hand model base on natural anatomy human hand appearance and its motion. This model is using super quadric surface to define a serial meatball to achieve good visual realism, and using texture mapping to represent the human hand skin, in order to enhance the realistic. And they are using CyberGlove dataglove as a virtual hand driven, and then the constraint of the hand model is no very important in this case. (Wan, Luo, Gao, & Peng, 2004)

#### **2.5 American Sign Language (ASL)**

American Sign Language (ASL, also Amslan obs,) is the dominant sign language of the Deaf community in the United States, in the English-speaking parts of Canada, and in parts of Mexico. It is a manual language or visual language, meaning that the information is expressed not with combinations of sounds but with combinations of handshakes, palm orientations, movements of the hands, arms and body, and facial expressions.(Wikipedia.com, 2006a) The American Sign Language Alphabet is a manual alphabet that augments the vocabulary of American Sign Language when spelling individual letters of a word is the preferred or only option, such as with proper names or the titles of works. It contains A-Z 26 letters and 0-9 ten numbers (Figure 2.5.1). Most of this gesture is still, these gestures are very easy represent by still image or 3D hand model. But for “J” and “Z”, they are a serial movement is hard

to simulate by still image.

American Sign Language alphabet

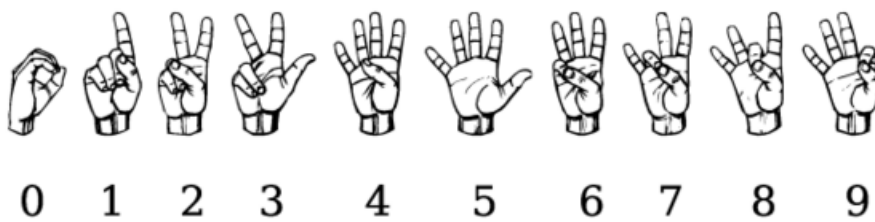
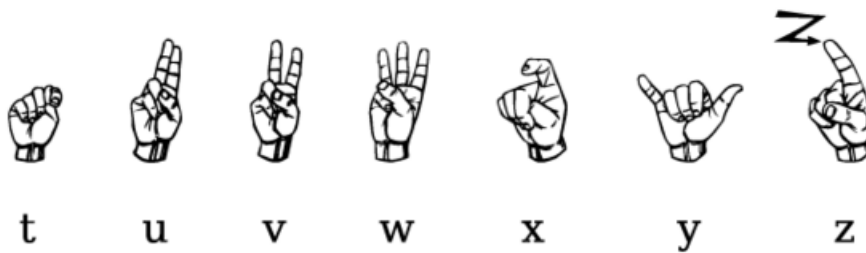
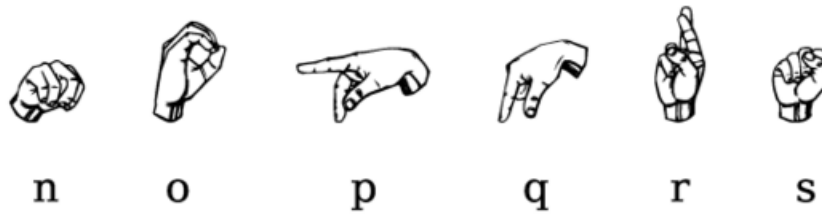
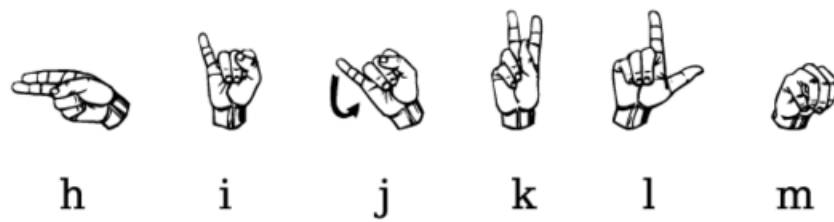
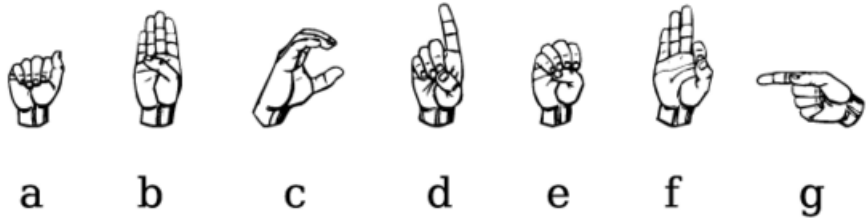


Figure 2.5.1: American Sign Language (ASL) Language alphabet

### 3.0 Methodology

The 3D hand models are being created by Povray language, and using 21 DOFs principle from Strenger.(Stenger, 2001) The main idea of these hand models is base on the articulate hierarchical of human hand, using Povray to create some basic objects, just as, sphere, cylinder and cone. Using cylinder or cone to simulate the finger bone and sphere to simulate the finger joints, through positioning and adjusting the basic objects, then a 3D articulate hand model can be create. To understand the Povray 3D modeler will be the foundation of the whole project.

#### 3.1 Povray

Using Povray to create 3D object is the base stone for the 3D hand modeling. As describe before, Povray is a “rendering engine”. It takes \*.pov file, render it and output an image file. \*.pov file is the source code file list of description of objects. First thing we need to know before start to create an object is, where the object will be place in three dimension world, such a method is called a coordinate system.

The coordinate system that POV-Ray uses, then, is called a three-dimensional (or 3D) Cartesian coordinate system. A quick graph looks like this:

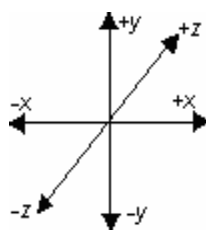


Figure 3.1.1: Cartesian coordinate system

The building blocks of all POV-Ray objects and scenes are called primitives. Primitives are objects that POV-Ray already knows about, and all you have to do is describe a few attributes. POV-Ray primitives are usually simple geometric shapes such as spheres, cubes, and cones. (The\_Online\_POV-Ray\_Tutorial\_ThinkQuest\_Team, 1996)

Describing primitives, in general, take this form in POV-Ray:

```
Object_Name {  
  Object_Parameters  
  Some_Simple_Attribute  
  Some_Other_Simple_Attribute  
  Some_Complex_Attribute {  
    Some_Attribute  
    Some_Other_Attribute  
  }  
}
```

For example

```
Sphere {<0,0,0> 1 pigment{color Red}}
```

This primitive means creating a sphere, where the sphere's centre will be  $x=0$ ,  $y=0$ ,  $z=0$ , and the radius is one in the coordinate system of Povray. And the color will be "Red". The defined of other primitives' objects are very similar like the example.

### 3.1.1 Camera and light

As mentioned before, Povray is ray-tracing software, so the first objects need to be created is the Camera and light. For example:

```
camera {  
  location <1,2,-10>  
  look_at <0,0,0>  
}
```

In this example, it defines a camera located at  $x=1$ ,  $y=2$ ,  $z=-10$  location, and look at the origin  $\langle 0, 0, 0 \rangle$ . According to the ray-tracing principle, it means that anything with a  $z$  coordinate less than  $-10$  will definitely be invisible—it will be behind the camera!

Similar to the camera the light can be defined as:

```
light_source {<0, 0, 10> color White}
```

It means defined a light at location  $x=0$ ,  $y=0$ ,  $z=10$  and light color will be white.

### 3.1.2 Cylinder:

The cylinder primitive creates a circular cylinder with the given characteristics. The syntax of the primitive is the following:

```
cylinder {  
  <center-1>, <center-2>, radius  
}
```

$\langle \text{center-1} \rangle$  and  $\langle \text{center-2} \rangle$  are vectors specifying the  $\langle x, y, z \rangle$  of the centers of the ends of the cylinder. Radius specifies the radius of the cylinder. Note that the sides of the cylinder will always be perpendicular to the ends. An example cylinder declaration:

```
cylinder {  
  <0, 0, 0>, <0, 1, 0>, 0.5  
  Pigment {color Yellow}  
}
```

This will create a circular cylinder that has the same height as diameter.

### 3.1.3 Cone

The cone primitive creates a cone with the given characteristics. It is similar to the cylinder, but two radii are specified instead of just one. Its syntax:

```
cone {  
  <center-1>, radius-1,  
  <center-2>, radius-2  
}
```

<center-1> and <center-2> are  $\langle x, y, z \rangle$  vectors for the centers of the two ends of the cone. radius-1 and radius-2 specify the radius of the cone at <center-1> and <center-2> respectively. Both radius-1 and radius-2 can be positive, negative, or zero. If one radius is zero, the object looks like a standard cone shape. If both radii are the same sign, the object looks like a cone with the tip cut off. If the radii are opposite signs, the object looks like two cones with their tips touching. An example cone:

```
cone {  
  <0, 0, 0>, 3,  
  <0, 3, 0>, 0  
  Pigment { color Blue }  
}
```

This will create a cone that is three units tall and six units in diameter at the base.

### 3.1.4 Plane

The plane primitive creates an infinite, flat plane of any orientation and any location.

The syntax of plane is as follows:

```
plane {  
  <normal>, offset  
}
```

<normal> is an <x, y, z> vector that describes the plane by giving a normal to its surface. By default, the plane goes through the origin. This is changed with the offset value. It is a float that translates the plane along its normal vector. It may be positive, negative, or zero, and can be used to position the plane anywhere in space. Here are some examples of planes:

```
plane {  
  y, 3  
}
```

This plane is parallel to the x-z coordinate plane. It extends infinitely in both the x and z directions. And there are some basic Transformation, such as translation, rotation, and scale.

## 3.2 Basic Transformation

Basic transformation can modify the object, these modifies can increase or decrease the size of object, rotate the object around x, y, z axis, or translate the object to new place.

### 3.2.1 Translation:

A translation is a transformation that moves an object relative to its current position. It is specified in POV-Ray by the phrase `translate <x, y, z>`. Translations are easy to visualize. Consider a cube sitting on the origin, like this:

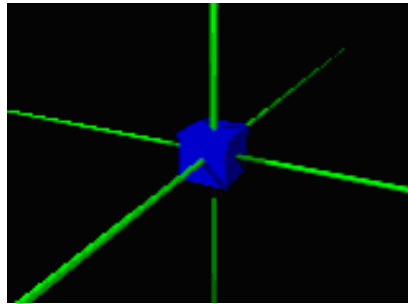


Figure 3.2.1.1: Original image.

Our camera is positioned so that the x axis increases to the right, the y axis increases upwards and the z axis increases towards us. A translation of `<-1,4,2>` results in the cube being moved left one unit, up four, and back two, like this:

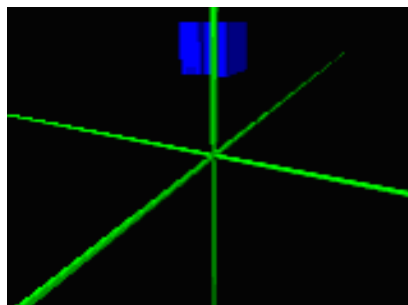


Figure 3.2.1.2: Translation.

### 3.3.2 Rotation

A rotation is a transformation that changes the orientation of an object (the way that it's facing). Rotations are the most complex of the transformations. They are specified to POV-Ray by the string `rotation <x, y, z>`, where x, y, and z are the number of

degrees (not radians) around the respective axis. Consider the original cube up above. A rotation of  $\langle 0, 0, 45 \rangle$  rotates the cube 45 degrees around the z axis, leaving us with a cube looking like this:

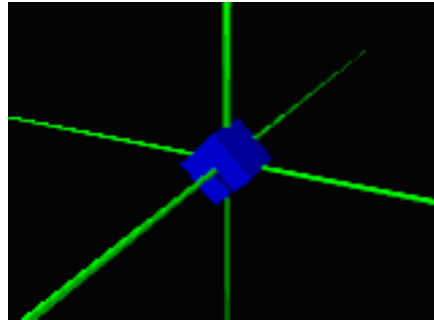


Figure 3.3.2: Rotation.

### 3.3.3 Scaling

The last translation you need to know about is scaling. Simply enough, scaling changes the size of the object with respect to its current size. Scaling is specified in POV-Ray via the string scale  $\langle x, y, z \rangle$ . The elements of the vector specify the how much to scale the shape with respect to the coordinate axis: a scale of 1.0 leaves the object the same, and a scale of 0.0 or less is invalid. Going back to our original cube, if we scaled the object with the string scale  $\langle 1, 4, 1 \rangle$ , we would get a result like this:

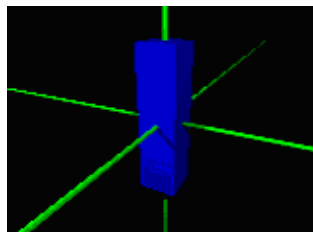


Figure 3.3.3: Scaling

Beside these objects above, there are some advanced objects that can be use to create

objects, such as quadrics, cubic, and polynomial. For this hand model creation, only the quadrics is useful.

### 3.3 Advanced features

Here are some advanced Povray features, which are useful for the hand model. They are “declare” syntax, “merge” syntax, Interaction operation and “quadric” syntax.

#### 3.3.1 “Declare” syntax.

POV-Ray provides a very powerful, very flexible way to create many similar objects with a statement called #declare. It can be use for create new type of object. And these object can be create by a serial simpler objects. After declaration, the new object can be reuse in whenever you like. The syntax will look as following:

```
#declare my_sphere =  
sphere {  
  <0, 0, 0>, 5  
  finish {  
    pigment rgbf <.5, .2, .4, .667>  
  }  
}
```

These codes declare a new object call “my\_sphere”. When you need to use this object, you just need to use code like these:

```
object {  
  my_sphere  
  translate <-2, 0, 0>  
}
```

Then you can use the new object and translate x axis decrease to left.

### 3.3.2 “Merge” syntax.

In order to simulate the hierarchical hand model in Povray, this “Merge” syntax is very important. “Merge” is used for simply takes a bunch of objects, and “sticks them together”. Or you can said them are group together. In this case, any transformation for this new grouped object will apply for the whole grouped. For example:

```
#define two_spheres =  
Merge {  
    sphere { <0, 0, 0>, 3 }  
    sphere { <-1, -5, -1>, 3 }  
}  
Object {  
    two_spheres  
    Pigment { color Pink }  
    Rotate <0, 180, 0>  
}
```

In this example, a new object “two\_spheres” has been declared and group together. Then, when the object “two\_spheres” is used, both two spheres will be assign to pink color and rotate 180° around y axis together.

### 3.3.3 Intersection

Much as a difference removes the insides of objects, an intersection removes the outsides of objects. The result of using the intersection operator is that the only thing remaining is the parts which all the objects within the operator had in common.

### Intersection



Figure 3.3.3: Intersection.(The\_Online\_POV-Ray\_Tutorial\_ThinkQuest\_Team, 1996)

#### 3.3.4 Quadric

A quadric is a second order polynomial. It can be used to create simple shapes like cylinders, cones, spheres, and hyperboloids. The only different between the quadric primitive and sphere primitive is POV-Ray has special optimizations for spheres primitive. But quadric primitive can some special objects.

The syntax for a quadric is as follows:

```
quadric {
  <a, b, c>,
  <d, e, f>,
  <g, h, i>,
  j
}
```

'a' through 'j' are all floats and correspond to the following second order polynomial equation:

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0$$

For example: this quadric create a sphere of radius 3 centered a <3,3,3>

```
quadric {
```

$$\begin{aligned} &\langle 1, -1, -1 \rangle, \\ &\langle 0, 0, 0 \rangle, \\ &\langle 0, 0, 0 \rangle, \\ &0 \\ &\} \end{aligned}$$

Quadric is complex, and some principle will be describe as following.

### 3.4 Super quadric

In mathematics a quadric, or quadric surface, is any D-dimensional hyper surface defined as the locus of zeros of a quadratic polynomial. In coordinates  $\{x_0, x_1, x_2, \dots, x_D\}$ , the general quadric is defined by the algebraic equation (Wikipedia.com, 2006b)

$$\sum_{i,j=0}^D Q_{i,j}x_i x_j + \sum_{i=0}^D P_i x_i + R = 0$$

Where Q is a D+1 dimensional matrix and P is a D+1 dimensional vector and R a constant. The values Q, P and R are often taken to be real numbers or complex numbers, but in fact, a quadric may be defined over any field.(Wikipedia.com, 2006a)

A second-order algebraic surface given by the general equation

$$ax^2 + by^2 + cz^2 + 2fyz + 2gzx + 2hxy + 2px + 2qy + 2rz + d = 0.$$

Quadratic surfaces are also called quadrics, and there are 17 standard-form types, which are ellipsoids, Cone, Cylinders, pairs of planes, and so on (Mathworld.wolfram.com, 1999)

#### 3.4.1 Ellipsoid

In mathematics, an ellipsoid is a type of quadric that is a higher dimensional analogue

of an ellipse. The equation of a standard ellipsoid in an x-y-z Cartesian coordinate system is

$$\frac{x^2}{w^2} + \frac{y^2}{h^2} + \frac{z^2}{d^2} = 1$$

Where a, b and c (the lengths of the three semi-axes) are fixed positive real numbers determining the shape of the ellipsoid. If two of those numbers are equal, the ellipsoid is a spheroid; if all three are equal, it is a sphere.

If we are using homogeneous coordinates to represent it, it will be a symmetric 4 x 4 matrix(Stenger, 2001):

$$Q = \begin{bmatrix} \frac{1}{w^2} & 0 & 0 & 0 \\ 0 & \frac{1}{h^2} & 0 & 0 \\ 0 & 0 & \frac{1}{d^2} & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

### 3.4.2 Cone

The equation of a standard implicit form of a cone aligned with the y axis is given by:

$$\frac{x^2}{w^2} - \frac{y^2}{h^2} + \frac{z^2}{d^2} = 0$$

And using homogeneous coordinates to represent it, it will be a symmetric 4 x 4 matrix(Stenger, 2001):

$$Q = \begin{bmatrix} \frac{1}{w^2} & 0 & 0 & 0 \\ 0 & -\frac{1}{h^2} & 0 & 0 \\ 0 & 0 & \frac{1}{d^2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

### 3.4.3 Cylinders

In mathematics, a cylinder is a quadric, i.e. a three-dimensional surface, with the following equation in Cartesian coordinates:

$$\left(\frac{x}{w}\right)^2 + \left(\frac{y}{h}\right)^2 = 1$$

This equation is for an elliptic cylinder, a generalization of the ordinary, circular cylinder ( $a = b$ ). Even more general is the generalized cylinder: the cross-section can be any curve.

The cylinder is a degenerate quadric because at least one of the coordinates (in this case  $z$ ) does not appear in the equation. The corresponding matrix is given by (Stenger, 2001):

$$Q = \begin{bmatrix} \frac{1}{w^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d^2} & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

### 3.4.4 Pair of planes

A pair of planes, which are parallel to the  $xz$  plane, the equation can be given by:

$$(y - y_0)(y - y_1) = 0$$

The corresponding matrix is given by(Stenger, 2001):

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{(y_0 + y_1)}{2} \\ 0 & 0 & 0 & 0 \\ 0 & -\frac{(y_0 + y_1)}{2} & 0 & y_0 y_1 \end{bmatrix}$$

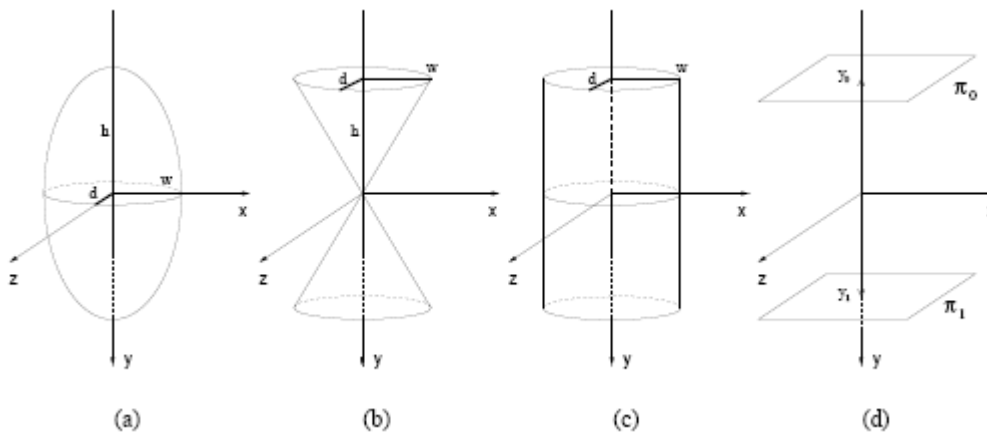


Figure 3.1: Examples of different quadric surfaces corresponding to the equations in the text: (a) ellipsoid, (b) cone, (c) elliptic cylinder, (d) pair of planes.

Notation: w=width, h=height, d=depth.(Stenger, 2001)

### 3.3 Implement quadric in Povray

The syntax for a quadric using in Povray is as follows:

```
Quadric {
  <a, b, c>,
  <d, e, f>,
  <g, h, i>,
  j
}
```

The letters in this syntax are corresponding to the following second order polynomial equation:

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0$$

As mention before, super quadric can create ellipsoid, cylinder, cone and so on, and their equation are easily transformed to the general equation.

According to the basic transformation in 3D coordinate system, a new position X' can be representing as following:

$$x' = x - x_0$$

$$y' = y - y_0$$

$$z' = z - z_0$$

If the ellipsoid is not on the origin points of the coordinate system,  $\langle x_0, y_0, z_0 \rangle$  represents the new center of the ellipsoid, then the equation will be:

$$\frac{(x - x_0)^2}{w^2} + \frac{(y - y_0)^2}{h^2} + \frac{(z - z_0)^2}{d^2} = 1$$

Then ellipsoid's equation can be:

$$(hdx)^2 + (wdy)^2 + (whz)^2 - 2x_0h^2d^2x - 2w^2d^2yy_0 - 2w^2h^2zz_0 + (hd x_0)^2 + (wdy_0)^2 + (whz_0)^2 - (whd)^2 = 0$$

As a result, square of 'hd' can use to represent as 'a' in the general equation, square of 'wd' can be representing as 'b' in general equation. Afterwards, we can create an ellipsoid very easy by know the center of the ellipsoid and its height, width, and depth.

Cone's equation can be:

$$(hdx)^2 - (ywd)^2 + (zwh)^2 - 2x_0h^2d^2x + 2y_0w^2d^2y - 2z_0w^2h^2z + (hdx_0)^2 - (wdy_0)^2 + (z_0wh)^2 = 0$$

Cylinder's equation can be:

$$(dx)^2 + (wz)^2 - 2x_0d^2x - 2z_0w^2z + (x_0d)^2 + (z_0w)^2 - (wd)^2 = 0$$

And pair of plane's equation can be:

$$y^2 - (y_1 + y_0)y + y_0y_1 = 0$$

After we know the relationship between the “Quadric” syntax in Povray and general equations of ellipsoid, cone, etc, then the quadric can be create like this:

```
Quadric{
  <36,9,4>,
  <0,0,0>,
  <-72,-18,0>,
  9
}
```

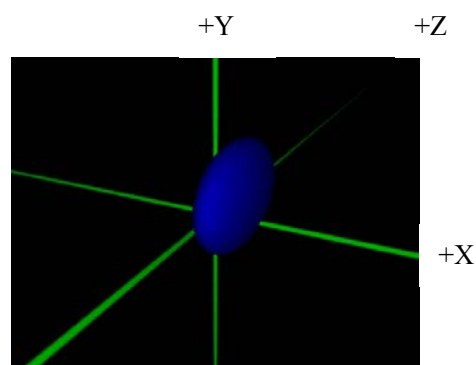


Figure 3.3.1: Example of quadric. The ellipsoid's center  $\langle x_0, y_0, z_0 \rangle$  know as  $\langle 1, 1, 0 \rangle$  and width(W)=1, height(H)=2, depth(D)=3.

## 4.0 Model implement

All of these three models can simulate 21 degree of freedoms (DOFs) of human hand movement. As shown in figure 2.3.1.2. The structure of these human hand models are base on the model create by Strenger(Stenger, 2001), a hierarchical with 21 degree of freedoms (DOFs). It used 4 for the motivation for each fingers, 5 for the pose of the thumb.

### 4.1 Hand Model 1 (Sphere & Cone)

This 3D human model is using a set of sphere, cone and cylinders to represent the anatomy of a real human hand. It basic on Stenger's 21 DOFs hand model using in Unscented Kalman filter(Stenger, 2001). The coordinate system of this model is starting from palm and ending at the tips, each element's position is basic on its pervious one, in order to similar the hierarchy of real human hand.

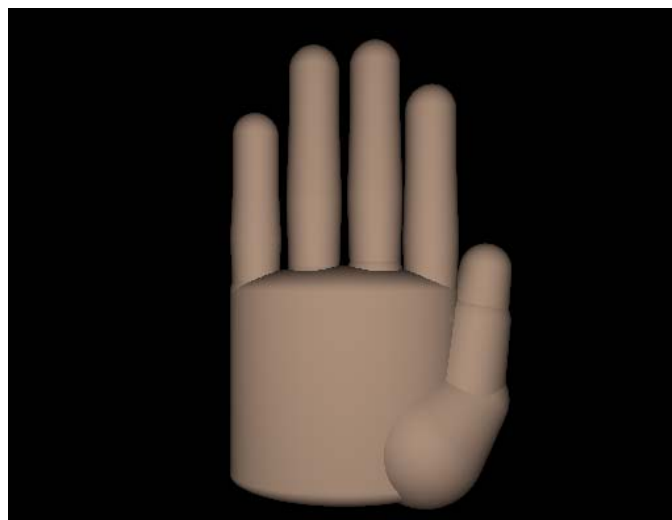


Figure 4.1.1: Hand model 1 creates by sphere, cone and cylinder.

The basic of the palm is modeled using a cylinder, its top and bottom closed by half

ellipsoids. There are 3 cones and 3 spheres for each fingers, include one sphere to represent finger tip, 3 cones to represent each phalanx, and they are connected by spheres, to represent the finger joints. The thumb is consists of 2 cones, three spheres, there is one sphere to represent MPJ, and one sphere are used to represent IJ, the cones are used to represent Metacarpal, Proximal Phalanx, and Thumb Distal Phalanx.

Through positioning the ellipsoid, adjusting their sizes and orientations, the model can be easily define to reach a good visual realism.

#### **4.2 Hand model 2 (Quadric Ellipsoid)**

This model is using a set of quadric ellipsoids to represent the anatomy of a real human hand. The coordinate system of this model is starting from palm and ending at the tips, each ellipsoid's position is basic on its pervious one, in order to similar the hierarchy of real human hand. The same as model one.

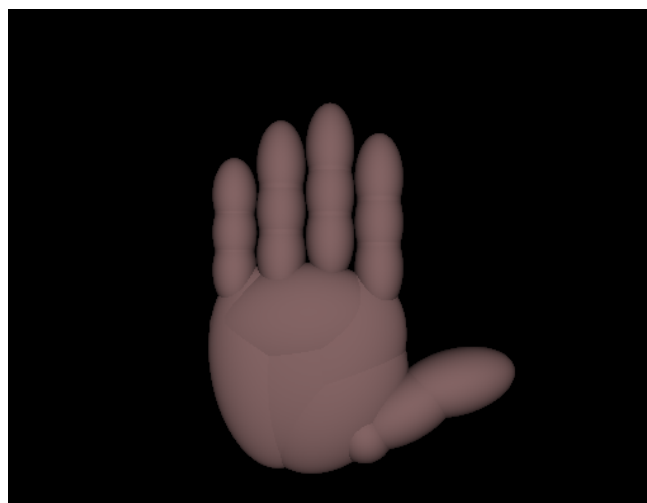


Figure 4.2.1: Hand model two creates by quadric ellipsoids.

The basic element of the palm is modeled using an ellipsoid, and there are 3 other ellipsoids are used to represent the detail of the palm, such as hand line. There are 6 ellipsoids for each fingers, include 3 ellipsoids to represent each phalanx, and they are connected by spheres, to represent the finger joints. The thumb is consists of 4 ellipsoids, there is 1 spheres represent TMJ, and the rest of the ellipsoids are used to represent Metacarpal, Proximal Phalanx, and Thumb Distal Phalanx.

Through positioning the ellipsoid, adjusting their sizes and orientations, the model can be easily define to reach a good visual realism.

In this model, the basic element is ellipsoid. These ellipsoids are created by “quadric” syntax in Povray. There is a function in Povray file using to create ellipsoid, which call basic function just input the location and detail of the ellipsoid, such as height, width and depth. Then the function will create ellipsoid as requirement.

### **4.3 Hand model 3 (Quadric Ellipsoid & Cone)**

This model is using a set of truncated quadrics to represent the anatomy of a real human hand. The coordinate system of this model is starting from palm and ending at the tips, each ellipsoid’s position is basic on its pervious one, in order to similar the hierarchy of real human hand. The same as model one.

The basic element of the palm is modeled using an ellipsoid, and there are 3 other

ellipsoids are used to represent the detail of the palm, such as hand line.

For each finger, there are 3 spheres and 3 cones, include one sphere for finger tip, 3 cones to represent each phalanx, and they are connected by spheres, to represent the finger joints. The thumb is consists of 2 cones, three spheres, there is one sphere to represent MPJ, and one sphere are used to represent IJ, the cones are used to represent Metacarpal, Proximal Phalanx, and Thumb Distal Phalanx.

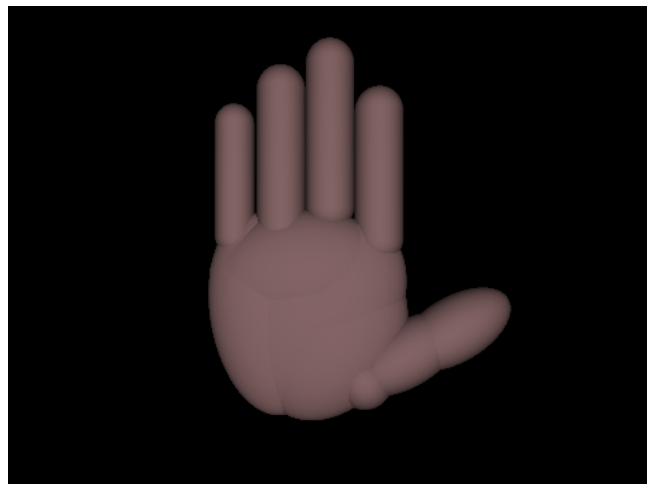


Figure 4.3.1: Hand model three creates by quadric ellipsoids, cones and cylinders.

Through positioning the ellipsoid, adjusting their sizes and orientations, the model can be easily define to reach a good visual realism.

In this model, the basic element is called “bone”, which included two spheres and connected with a cylinder. The “bone” is created by “quadric” syntax in Povray. There is a function in Povray file using to create this “bone” element, just input the location and detail of the ellipsoid or cones, such as height, width and depth. Then the function

will create a finger bone as requirement.

#### 4.4 Hierarchical Articulate of Three Models

In order to simulate the hierarchical movement of real human hand in Povray, the model uses “Merge” syntax to do it. And three models are the same.

For each finger, grouped distal finger-bone bone and Distal Interphalangeal Joint (DIJ) sphere together and give them a name called “finger\_top”, then distal finger-bone and Distal Interphalangeal Joint (DIJ) sphere and movement together, when the Distal Interphalangeal Joint (DIJ) do any movement.

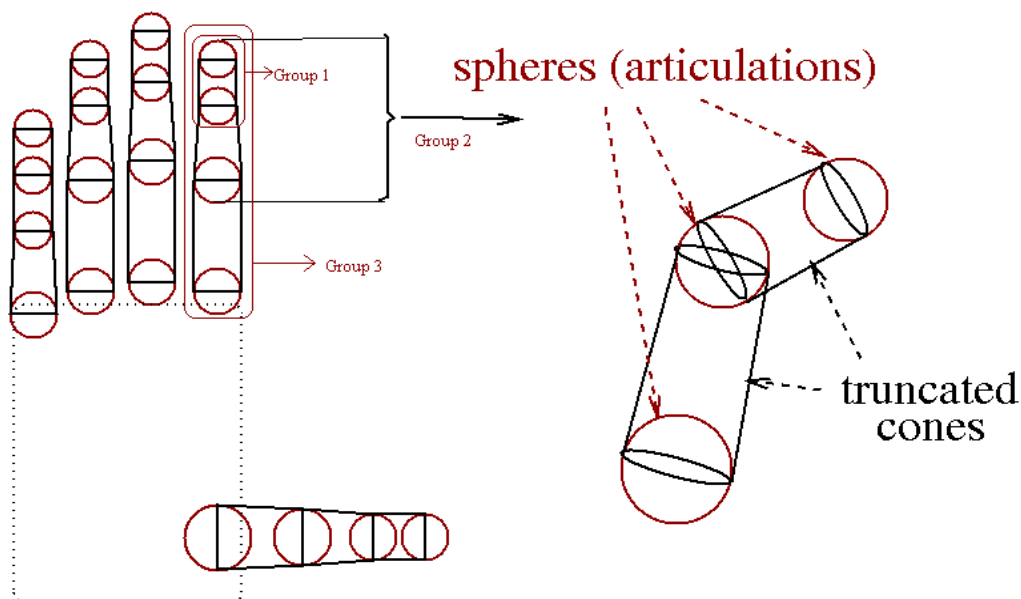


Figure 4.4.1: Articulation of hand model. (Delamarre & Faugeras, 2004)

Similar, grouped “finger\_top” and middle finger-bone and Proximal Interphalangeal Joint (PIJ) sphere together called “finger\_middle”, then all of there grouped object

will be movement together, when the Proximal Interphalangeal Joint (PIJ) do any movement. The last step for the hierarchical finger is grouped the proximal finger-bone and proximal sphere and “finger\_middle” and “finger\_top” together, then when the Metacarpophalangeal Proximal Joint (MPJ) sphere move, the whole finger will move.

Because these three models are create in articulate hierarchical structure, then once the sphere, which uses to represent joint, rotate some degree, consequently, the other part of fingers base on this joint will rotate together. For example using template model 3 (quadric cone), the original hand model look like figure 4.4.2(a), the index finger of the model is straight up. After change the 3 knuckles of index finger, to make them rotate around x axis in 70 degrees, then you can get figure 4.4.2(b).

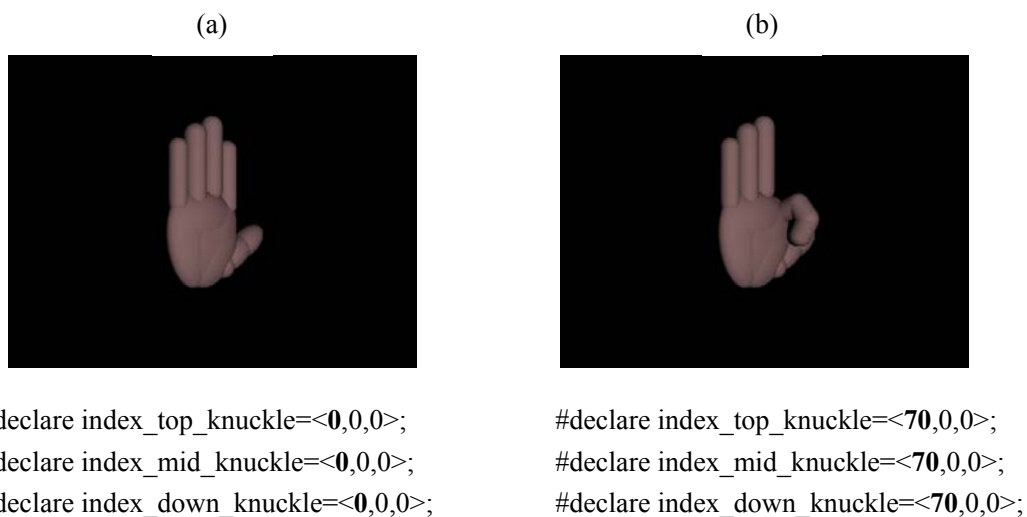


Figure 4.4.2: Index Finger rotation.

In this case, any finger movement can be control by the knuckle of the finger. Because three models have the same structure of finger, then all of them can be control in

similar way.

#### **4.5 Create Gesture Image by Models**

As mention above, hand model can be adjust into any gesture by change the finger knuckle rotation degree. But adjust very fingers for one gesture every time; in order to render an image is very annoying. So, saving fingers rotation degree for one or more gesture into a TXT file, a C++ program combine this gesture information and a hand Model together, and output a gesture Povray file, can save a lot of time. (Appendix 4 & 5)

This C++ program will collect the gesture information from TXT file and put this information with hand model template together, and output a Povray file. Render this Povray file by Povray, and then an ASL gesture still image can be created.

## 5.0 Result

These three models are using Povray language to create. The basic articulate structure is very similar, so the movements of hand models are the same. But there are some differences between them.

### 5.1 Model one (Sphere & Cone)

This hand model as figure 3.1.1 creates by cone, sphere and cylinder. It is suitable for real-time hand tracking; because it is use the simplest objects in Povray and these objects have been optimal by the Povray developers. So this model can be render really fast.

But it is not good be using at computer graphic or computer animation, because it is not realistic enough, the palm and fingers didn't look real and lack of detail. The surfaces of these simplest objects are fixing. There is no way to use these simplest to create other kind of surfaces. For example, "Sphere" syntax only can create sphere, no way to create ellipsoid using "sphere" syntax itself. On the other hand, the basic of this model are optimal objects, they can be modified, like using rotation or scaling, but it will take more time in rendering process. In this case, any detail update of this model will relate to the whole model recreation.

The advantage of this model is speed, and it is easy to create. The disadvantage of this

model is unrealistic, hard to modified and update.

## **5.2 Model Two (Quadric Ellipsoid)**

This model is created by ellipsoids, and these ellipsoids are creating by “quadric” syntax in Povray language. Using quadric has an advantage on the surface creation. It is easy to create a serial quadric surface, sphere, ellipsoid, cone, cylinder, so on. In this model only ellipsoid is used, ellipsoid is the basic object of this model, and can be modifier easily. As a result, this model hand provides enough detail of human hand, such as, hand line, knuckle and so on. The render speed of this model is slower than model one. It may use in real-time hand tracking, which requires the realistic hand model represent in program. And some computer graphic or animation also may use it.

Meanwhile, the model can be improved in several ways. First, the finger top is to sharp cause the model doesn't looks well. Second, the edge of four fingers displays as curve, but as real human hand it doesn't look like that. As real human hand, the finger top looks like a round sphere more than a sharp ellipsoid, and the edge of fingers should display as line not curve.

The advantages of this model are acceptable render speed, easy to modifier and provide enough detail as a human hand. The disadvantage is low display quality of four fingers.

### **5.3 Model Three (Quadric Ellipsoid & Cone)**

As description before, this model is update from model two. The palm and thumb didn't do any modification, but four fingers are changed from ellipsoids to spheres and cones. The reason of this change is that the edge of these fingers can be displayed in line, not a curve. Therefore, the display quality has significant improvement. It looks real like a human hand without a skin. In modification aspect, this model is using quadric as a basic object, so it is easy to change the basic object to other kind of object, which can be create by quadric in Povray. Computer graphic and animation can use this realistic hand model.

But as the display quality improved, the render speed decrease rapidly. Thereby, this model can not be used in real-time hand tracking. Realistic is this model's advantage, and the slow render speed is its disadvantage.

### **5.4 Feature Work**

All of these three models are the skeletal models; they just provide the basic articulate idea of 3D human hand model. The realistic of these models is far away from professional computer graphics or computer animation quality, but there is a way to fix this problem, texture. Using texture properly can significantly change the display quality and fix the unrealistic problem. To illustrate, this example is from State Key Lab of CAD & CG ZheJiang University in China.



(a) Shading display



(b) Textured display

Figure 4.4: The Textured Hand (Wan, Luo, Gao, &amp; Peng, 2004)

Texture changed the shading display 3D hand model to a real old man hand. It is very useful in some case, which requires very high quality of 3D hand model.

## 5.5 Summary

The advantage and disadvantage of three models are as following:

	Quality	Modification	Render Speed
Model 1 (Sphere and cone)	Bad	Hard	High
Model 2 (Quadric Ellipsoid)	OK	Easy	OK
Model 3 (Quadric Ellipsoid & Cone)	Good	Easy	Low

To sum up, each model has its own advantage, it cause the model has it own useful environment. Model one, which is using optimal sphere and cone, is suitable for the real-time hand movement tracking. Model two, which is using quadric ellipsoid, is suitable for real-time hand tracking or computer graphic/animation. Model three,

which is using quadric ellipsoid, cone and cylinder, is only suitable for the computer graphic or use for training the hand tracking system. Because all of three models are skeletal models, the realistic is not very high. But after using texture, the virtual hand will be enhanced by skin textures. Then usability of these models will be wider in the feature.

## 5.0 Conclusions

Human-computer interaction is still in its infancy. Hand tracking technology allows user using the one of the most natural interpersonal language (body language or Sign language) to communicate with computer.

This paper is using ray-tracing language Povray to create a 3D hand model template. These models are simulating the articulate and hierarchical of real human hand, they are skeletal models base on Jintae Lee and Tosiyasu L. Kunii's 27 DOFs Hand model created in 1995. (Jintae & Kunii, 1995). Each of these model content 21 DOFs, and simulate most of the gesture in American Sign Language (ASL). Model one is created by basic objects, model two is created quadric ellipsoids, and model three is created by truncated quadric. Each model has its own advantage. Model one suitable for real-time hand tracking, model two has balance between the displays quality and renders speed. Model three has good visual realism. There is a C++ program can use these 3D hand model template create a serial still image for training the hand tracking system, or directly use this model in hand tracking system, or other purposes.

## References

- Albrecht, I., Haber, J., & Seidel, H.-P. (2003). *Construction and animation of anatomically based human hand models*. Paper presented at the Conference Name|. Retrieved Access Date|. from URL|.
- C.Wren, A.Azarbayejani, T.Darrell, & A.Pentland. (1996). *Pfinder:Real\_Time Tracking of the human Body* Paper presented at the Int'l conf. Automatic Face and Gesture Recognition, Killington.
- D.Thompson. (1981). Biomechanics of the hand. *Perspectives in Computing, 1*, 12-19.
- D.Thompson, W.Buford, L.Myers, D.Giurintano, & J.Brewer. (August 1988). A hand biomechanics workstation. *Compuer Graphics, 22*, 335-343.
- Delamarre, Q., & Faugeras, O. (2004). Finding pose of hand in video images: a stereo-based approach. from <http://www-sop.inria.fr/robotvis/personnel/qdelam/ArticleFG98html/node2.html>
- E.Catmull. (1972). *A system for computer-generated movies*. Paper presented at the ACM Annual Conf.
- Edward, A. (1996). *Interactive computer graphics: a top-down approach with OpenGL*: Addison-Wesley Longman Publishing Co., Inc.
- Guan, H., Chua, C.-S., & Ho, Y.-K. (2001). *3D hand pose retrieval from a single 2D image*.
- H.Rijpkema, & M.Girard. (1991). *Computer- animation of knowledge-based human grasping,*” *Computer Gruphics*.

- J.Floey, Dam, A. V., S.Feiner, & J.Hughes. (1990). *Computer Graphics: Principles and Practice*.: MA: addison-Wesley Publishing Company.
- J.Gourret, Thalmann, N. M., & D.Thalmann. (July 1989). *Simulation of object and skin deformations in a grasping task*. Paper presented at the Computer Graphics,.
- J.J.Kuch, & T.S.Huang. (1994a). *Human computer interaction via the human hand: a hand model*.
- J.J.Kuch, & T.S.Huang. (1994b). *Virtual Gun: A Vision Based Huamn Computer Interface Using the Human Hand*. Paper presented at the Proc. IAPR Workshop on Mach.Vision App, Tokyo.
- Jintae, L., & Kunii, T. L. (1995). Model-based analysis of hand posture. *Computer Graphics and Applications, IEEE, 15(5), 77-86*.
- Mathworld.wolfram.com. (1999). Quadratic Surface. from <http://mathworld.wolfram.com/QuadraticSurface.html>
- N.Badler, & M.Morris. (1982). *Modeling flexible articulated objects*. Paper presented at the Computer Graphics.
- N.Magnenat-Thalmann, & D.Thalmann. (1990). *Computer Animation: Theory and Pracice*. Paper presented at the Springer-Verlag, New York.
- N.Magnenat-Thalmann, R.Laperriere, & D.Thalmann. (1988). *Joint-dependent local deformations for hand animation and objcct grasping*. Paper presented at the Graphics Intvrface.
- Pavlovic, V. I., Sharma, R., & Huang, T. S. (1997). Visual interpretation of hand gestures for human-computer interaction: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 19(7), 677-695*.

R.Koch. (1993). Dynamic 3-D scene analysis through synthesis feedback control. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6), 556-568.

Stenger, B. (2001). *3D Model-Based Hand Tracking*. University of Cambridge.

The\_Online\_POV-Ray\_Tutorial\_ThinkQuest\_Team. (1996). The Online POV-Ray Tutorial: Introduction to POV-Ray and Ray-tracing. from <http://library.advanced.org/3285/>

Vladimir I. Pavlovic, R. S., Thomas s.Huang,. (July 1997). *visual interpretation of hand gestrues for human-computer interaction: a review*.

W.Buford, L.Myers, & A.Hollister. (November 1989). *3-D compuer graphics simulation of thumb joint kinematics*. Paper presented at the Proceedings Annual International Conference IEEE Engineering in Medicine and Biology Society.

W.Cooney, M.Lucca, E.Chao, & Inscheid, R. I. (December 1981). The Kinesiology of the thumber trapeziometacarpal joint. *the Journal of Bone And Joint Surgery*, 63-A, 1371-1381.

Wan, H., Luo, Y., Gao, S., & Peng, Q. (2004). *Realistic virtual hand modeling with applications for virtual grasping*. Paper presented at the Conference Name|. Retrieved Access Date|. from URL|.

Wikipedia.com. (2006a). Hand. from <http://en.wikipedia.org/wiki/Hand>

Wikipedia.com. (2006b). Quadric. from <http://en.wikipedia.org/wiki/Quadric>

## Appendix A Model Template One Source Code

```
#include "colors.inc"
#include "textures.inc"
#include "shapes.inc"

##declare camera_location=<0,20,-20>; //look from back
##declare camera_location=<30,6,0>; //look from little finger.
#declare camera_location=<0,8,25>; //look from front.
##declare camera_location=<-20,6,0>; //look from thumb finger
##declare camera_location=<0,20,0>; //look from top

#declare camera_lookat=<0,8,0>;
camera {
    location camera_location
    look_at camera_lookat
}

#declare light_location=camera_location;

light_source {light_location color White}
//=====

//=====

//index
#declare finger_mpj_x=-3.4;
#declare finger_mpj_y=6.5;
#declare finger_mpj_z=0;

#declare finger_radius=1;

#declare l1=3;
#declare l2=3;
#declare l3=2;

#declare index_f_top=
merge{
    sphere{<finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>,finger_radius}

cone{<finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj
```

```

_y+11+12+13,finger_mpj_z>,finger_radius}
  sphere{<finger_mpj_x,finger_mpj_y+11+12+13,finger_mpj_z>,finger_radius}
}

#declare index_f_mid=
merge{
  sphere{<finger_mpj_x,finger_mpj_y+11,finger_mpj_z>,finger_radius+0.1}

  cone{<finger_mpj_x,finger_mpj_y+11,finger_mpj_z>,finger_radius+0.1,<finger_mpj_x,finger_m
pj_y+11+12,finger_mpj_z>,finger_radius}
  merge{
    index_f_top
    translate <-finger_mpj_x,-(finger_mpj_y+11+12),finger_mpj_z>
    rotate index_top_r
    translate <finger_mpj_x,finger_mpj_y+11+12,finger_mpj_z>
  }
  pigment{Flesh}
}

#declare index_f=
merge{
  sphere{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius+0.1 }

  cone{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj_y+11,
finger_mpj_z>,finger_radius+0.1}
  merge{
    index_f_mid
    translate <-finger_mpj_x,-(finger_mpj_y+11),finger_mpj_z>
    rotate index_mid_r
    translate <finger_mpj_x,finger_mpj_y+11,finger_mpj_z>
  }
  pigment{Flesh}
}

merge{
  index_f
  translate <-finger_mpj_x,-finger_mpj_y,finger_mpj_z>
  rotate index_down_r
  translate <finger_mpj_x,finger_mpj_y,finger_mpj_z>
  pigment{Flesh}
}

//=====middle=====

```

```

#declare finger_mpj_x=-1.2;
#declare finger_mpj_y=7.7;
#declare finger_mpj_z=0;

#declare finger_radius=1;

#declare l1=3;
#declare l2=3;
#declare l3=2;

#declare index_f_top=
merge{
  sphere{<finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>,finger_radius}

  cone{<finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj
_y+l1+l2+l3,finger_mpj_z>,finger_radius}
  sphere{<finger_mpj_x,finger_mpj_y+l1+l2+l3,finger_mpj_z>,finger_radius}
}

#declare index_f_mid=
merge{
  sphere{<finger_mpj_x,finger_mpj_y+l1,finger_mpj_z>,finger_radius+0.1}

  cone{<finger_mpj_x,finger_mpj_y+l1,finger_mpj_z>,finger_radius+0.1,<finger_mpj_x,finger_m
pj_y+l1+l2,finger_mpj_z>,finger_radius}
  merge{
    index_f_top
    translate <-finger_mpj_x,-(finger_mpj_y+l1+l2),finger_mpj_z>
    rotate middle_top_r
    translate <finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>
  }
  pigment{Flesh}
}

#declare index_f=
merge{
  sphere{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius+0.1 }

  cone{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj_y+l1,
finger_mpj_z>,finger_radius+0.1}
  merge{
    index_f_mid
    translate <-finger_mpj_x,-(finger_mpj_y+l1),finger_mpj_z>
    rotate middle_mid_r

```

```

        translate <finger_mpj_x,finger_mpj_y+11,finger_mpj_z>
    }
    pigment{Flesh}
}

merge{
    index_f
    translate <-finger_mpj_x,-finger_mpj_y,finger_mpj_z>
    rotate middle_down_r
    translate <finger_mpj_x,finger_mpj_y,finger_mpj_z>
    pigment{Flesh}
}

//=====ring=====
#declare finger_mpj_x=1.2;
#declare finger_mpj_y=7.5;
#declare finger_mpj_z=0;

#declare finger_radius=1;

#declare l1=3;
#declare l2=3;
#declare l3=2;

#declare index_f_top=
merge{
    sphere{<finger_mpj_x,finger_mpj_y+11+l2,finger_mpj_z>,finger_radius}

    cone{<finger_mpj_x,finger_mpj_y+11+l2,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj
_y+11+l2+l3,finger_mpj_z>,finger_radius}
    sphere{<finger_mpj_x,finger_mpj_y+11+l2+l3,finger_mpj_z>,finger_radius}
}

#declare index_f_mid=
merge{
    sphere{<finger_mpj_x,finger_mpj_y+11,finger_mpj_z>,finger_radius+0.1}

    cone{<finger_mpj_x,finger_mpj_y+11,finger_mpj_z>,finger_radius+0.1,<finger_mpj_x,finger_m
pj_y+11+l2,finger_mpj_z>,finger_radius}
    merge{
        index_f_top
        translate <-finger_mpj_x,-(finger_mpj_y+11+l2),finger_mpj_z>
        rotate ring_top_r
        translate <finger_mpj_x,finger_mpj_y+11+l2,finger_mpj_z>

```

```

    }
    pigment{Flesh}
}

#declare index_f=
merge{
    sphere{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius+0.1 }

cone{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj_y+11,
finger_mpj_z>,finger_radius+0.1}
    merge{
        index_f_mid
        translate <-finger_mpj_x,-(finger_mpj_y+11),finger_mpj_z>
        rotate ring_mid_r
        translate <finger_mpj_x,finger_mpj_y+11,finger_mpj_z>
    }
    pigment{Flesh}
}

merge{
    index_f
    translate <-finger_mpj_x,-finger_mpj_y,finger_mpj_z>
    rotate ring_down_r
    translate <finger_mpj_x,finger_mpj_y,finger_mpj_z>
    pigment{Flesh}
}

//=====pinky=====
#declare finger_mpj_x=3.5;
#declare finger_mpj_y=7;
#declare finger_mpj_z=0;

#declare finger_radius=0.9;

#declare l1=2;
#declare l2=2;
#declare l3=2;

#declare index_f_top=
merge{
    sphere{<finger_mpj_x,finger_mpj_y+11+l2,finger_mpj_z>,finger_radius}

cone{<finger_mpj_x,finger_mpj_y+11+l2,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj
_y+11+l2+l3,finger_mpj_z>,finger_radius}

```

```

    sphere{<finger_mpj_x,finger_mpj_y+l1+l2+l3,finger_mpj_z>,finger_radius}
}

#declare index_f_mid=
merge{
    sphere{<finger_mpj_x,finger_mpj_y+l1,finger_mpj_z>,finger_radius+0.1}

cone{<finger_mpj_x,finger_mpj_y+l1,finger_mpj_z>,finger_radius+0.1,<finger_mpj_x,finger_m
pj_y+l1+l2,finger_mpj_z>,finger_radius}
    merge{
        index_f_top
        translate <-finger_mpj_x,-(finger_mpj_y+l1+l2),finger_mpj_z>
        rotate pinky_top_r
        translate <finger_mpj_x,finger_mpj_y+l1+l2,finger_mpj_z>
    }
    pigment{Flesh}
}

#declare index_f=
merge{
    sphere{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius+0.1 }

cone{<finger_mpj_x,finger_mpj_y,finger_mpj_z>,finger_radius,<finger_mpj_x,finger_mpj_y+l1,
finger_mpj_z>,finger_radius+0.1}
    merge{
        index_f_mid
        translate <-finger_mpj_x,-(finger_mpj_y+l1),finger_mpj_z>
        rotate pinky_mid_r
        translate <finger_mpj_x,finger_mpj_y+l1,finger_mpj_z>
    }
    pigment{Flesh}
}

merge{
    index_f
    translate <-finger_mpj_x,-finger_mpj_y,-finger_mpj_z>
    rotate pinky_down_r
    translate <finger_mpj_x,finger_mpj_y,finger_mpj_z>
    pigment{Flesh}
}

//=====plam=====

```

```

#declare plam_y=7.0;
#declare plam_z=2.0;
#declare plam_x=4.5;

merge{
  cylinder {
    <0, 0, 0>, <0, plam_y, 0>, plam_z
    scale <plam_x/2,0,0>
    pigment { color Flesh }
  }

  sphere{
    <0,0,0> 1
    scale <plam_x,0,2>
    pigment {color Flesh}
  }
  /*sphere{
    <4,plam_z/2,0> 1
    scale <0,4.9,1.2>
    translate <0,-.7,0>
    //pigment {color Flesh}
  } */

  sphere{
    <0,plam_y,0> 1
    scale <plam_x,0,2>
    pigment {color Flesh}
  }
  pigment {color Flesh}
}

//=====thumb=====
#declare thumb_top=
object{
  Round_Cone2_Union(<0,7,0>, 1.2, <0,9,0>, 1) //include in shapes.inc
file.
  //translate <-3.6,-.9,0>
  rotate <0,0,10>
  translate <-3.6,-.9,0>
  //pigment {color Flesh}
}

#declare thumb_mid=
merge{

```

```
object{thumb_top}
object{
    Round_Cone2_Union(<0,4.5,0>, 1.5,<0,6,0>, 1.3)
    rotate <0,0,10>
    translate <-3.6,-.9,0>
    //pigment {color Flesh}
}
//translate <3.6,.9,0>
//rotate <0,0,30>
// translate <-3.6,-.9,0>
}
```

```
merge{
object{thumb_mid}
object{
    Round_Cone2_Union(<0,2,0>, 2.0, <0,4.5,0>,1.6)
    rotate <0,0,70>
    translate <-1.5,0,0>
    //pigment {color Red}
}
//translate <1.5,0,0>
//rotate <0,0,0>
//translate <-1.5,0,0>
pigment {color Flesh}
}
```

## Appendix B Model Template Two Source Code

```
//quadric_1 (ellipsoid)

#include "colors.inc"
#include "textures.inc"
#include "shapes.inc"

##declare camera_location=<0,10,-10>; //look from back
##declare camera_location=<30,6,>; //look from little finger.
#declare camera_location=<0,6,20>; //look from front.
##declare camera_location=<-20,6,0>; //look from thumb finger
##declare camera_location=<0,20,0>; //look from top

#declare camera_lookat=<0,6,0>;
camera {
    location camera_location
    look_at camera_lookat
}

##declare light_location= <0,6,20>;
#declare light_location=camera_location;

light_source {light_location color White}

#macro basic(x0,y0,z0,WX,HY,DZ) //ellipsoid H for y axis, w for x axis, d for z axis

    #local aa = HY*HY*DZ*DZ;
    #local bb = WX * WX * DZ * DZ;
    #local cc = WX * WX * HY * HY;
    #local gg = -(2 * x0 * HY * HY * DZ * DZ);
    #local hh = -(2 * y0 * WX * WX * DZ * DZ);
    #local ii = -(2 * z0 * WX * WX * HY * HY);
    #local jj = (HY * HY * DZ * DZ * x0 * x0) + (WX * WX * DZ * DZ * y0 * y0) + (WX * WX
    * HY * HY * z0 * z0) - (WX * WX * HY * HY * DZ * DZ);

    quadric {<aa,bb,cc >,<0,0,0>,<gg,hh,ii>,jj}
#end

#declare knuckle_0=.5;
```

```

//=====
//index finger
#declare f_x=-1.9;
#declare f_y=5.8;
#declare f_z=0.0;

#declare f_top_r=index_top_knuckle;
#declare f_mid_r=index_mid_knuckle;
#declare f_down_r=index_down_knuckle;

#declare f_size_wx=1;
#declare f_size_hy=1.8;
#declare f_size_dz=1;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

    object{ basic(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)}

}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    object{ basic(f_x,f_y+2*f_size_hy+knuckle,f_z,knuckle,knuckle,knuckle)

}
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{basic(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy,f_size_dz)

}

}

```

```

#declare finger_mid_Knuckle=
merge{
  object{finger_mid}
  object{ basic(f_x,(f_y+f_size_hy+knuckle),f_z,knuckle,knuckle,knuckle)
  }
  translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
  rotate f_mid_r
  translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

```

```

#declare finger_down=
merge{
  object{finger_mid_Knuckle}
  object{basic(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)

  }

}

```

```

merge{
  object{finger_down}
  object{basic(f_x,f_y,f_z,knuckle_0,knuckle_0,knuckle_0+.3)

  }pigment {color Pink}
  translate<-f_x,-f_y,-f_z>
  rotate f_down_r
  translate<f_x,f_y,f_z>
}

```

```

//=====

```

```

//middle finger
#declare f_x=0.1;
#declare f_y=7.0;
#declare f_z=0.0;

#declare f_top_r=middle_top_knuckle;
#declare f_mid_r=middle_mid_knuckle;
#declare f_down_r=middle_down_knuckle;

#declare f_size_wx=1;
#declare f_size_hy=1.8;
#declare f_size_dz=1;

```

```

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

    object{ basic(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
            pigment {color Pink}
        }
}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    object{ basic(f_x,f_y+2*f_size_hy+knuckle,f_z,knuckle,knuckle,knuckle)
            pigment {color Pink}
        }
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{basic(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy,f_size_dz)
            pigment {color Pink}
        }
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    object{ basic(f_x,(f_y+f_size_hy+knuckle),f_z,knuckle,knuckle,knuckle)pigment {color
Pink}}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{

```

```

    object{finger_mid_Knuckle}
    object{basic(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
        pigment {color Pink}
    }
}

merge{
    object{finger_down}
    object{basic(f_x,f_y,f_z,knuckle_0,knuckle_0,knuckle_0)
        pigment {color Pink}
    }
    translate<-f_x,-f_y,-f_z>
    rotate f_down_r
    translate<f_x,f_y,f_z>
}

//=====
//Ring finger
#declare f_x=2.1;
#declare f_y=6.3;
#declare f_z=0.0;

#declare f_top_r=ring_top_knuckle;
#declare f_mid_r=ring_mid_knuckle;
#declare f_down_r=ring_down_knuckle;

#declare f_size_wx=1;
#declare f_size_hy=1.8;
#declare f_size_dz=1;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{
    object{ basic(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
        pigment {color Pink}
    }
}

#declare finger_top_Knuckle=
merge{

```

```

    object{finger_top}
    object{ basic(f_x,f_y+2*f_size_hy+knuckle,f_z,knuckle,knuckle,knuckle)
            pigment {color Pink}
    }
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{basic(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy,f_size_dz)
            pigment {color Pink}
    }
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    object{ basic(f_x,(f_y+f_size_hy+knuckle),f_z,knuckle,knuckle,knuckle)pigment {color
Pink}}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{basic(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
            pigment {color Pink}
    }
}

merge{
    object{finger_down}
    object{basic(f_x,f_y,f_z,knuckle_0,knuckle_0,knuckle_0)
            pigment {color Pink}
    }
    translate<-f_x,-f_y,-f_z>
}

```

```

    rotate f_down_r
    translate<f_x,f_y,f_z>
}

//=====
//Little finger
#declare f_x=4.0;
#declare f_y=6.0;
#declare f_z=0.0;

#declare f_top_r=little_top_knuckle;
#declare f_mid_r=little_mid_knuckle;
#declare f_down_r=little_down_knuckle;

//
#declare f_size_wx=.9;
#declare f_size_hy=1.5;
#declare f_size_dz=.9;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

    object{ basic(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
            pigment {color Pink}
        }
}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    object{ basic(f_x,f_y+2*f_size_hy+knuckle,f_z,knuckle,knuckle,knuckle)
            pigment {color Pink}
        }
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{

```

```

    object{finger_top_Knuckle}
    object{basic(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy,f_size_dz)
        pigment {color Pink}
    }
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    object{ basic(f_x,(f_y+f_size_hy+knuckle),f_z,knuckle,knuckle,knuckle)pigment {color
Pink}}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{basic(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy,f_size_dz)
        pigment {color Pink}
    }
}

merge{
    object{finger_down}
    object{basic(f_x,f_y,f_z,knuckle_0,knuckle_0,knuckle_0)
        pigment {color Pink}
    }
    translate<-f_x,-f_y,-f_z>
    rotate f_down_r
    translate<f_x,f_y,f_z>
}

//=====
//plam
#declare f_x=0.9;
#declare f_y=3.5;
#declare f_z=0.0;
union{

```

```

//sphere{<0,0,0> 1 pigment {color Green}}
object{
    basic(f_x,f_y,f_z,4,4,1.3)
    //pigment {color Pink}

}
//=====
#declare f_x=2.7;
#declare f_y=3.6;
#declare f_z=0.0;

object{
    basic(0,0,0,2.3,4.1,1.2)
    rotate<0,0,-10>
    translate <f_x,f_y,f_z>
    //pigment {color Pink}

}
//
#declare f_x=1.3;
#declare f_y=5.3;
#declare f_z=0.0;
object{
    basic(0,0,0,3.75,2.5,1.35)
    rotate<0,0,-5>
    translate <f_x,f_y,f_z>
    //pigment {color Pink}

}

#declare f_x=-0;
#declare f_y=3.8;
#declare f_z=0.0;
object{
    basic(0,0,0,2.7,3.9,1.3)
    rotate<0,0,30>
    translate <f_x,f_y,f_z>
    //pigment {color Pink}

}
//=====
//thumb base
#declare f_x=-0.0;
#declare f_y=2.6;

```

```

#declare f_z=0.0;

object{
    basic(0,0,0,2.7,3.6,1.32)
    rotate<0,0,40>
    translate <f_x,f_y,f_z>

}
pigment {color Pink}
}
//=====
//thumb
##declare t_top_r=thumb_top_knuckle;
##declare t_mid_r=thumb_mid_knuckle;
##declare t_down_r=thumb_down_knuckle;

#declare t_top_r=<0,0,0>;
#declare t_mid_r=<0,0,0>;
#declare t_down_r=<0,0,0>;

#declare foref_x=-5.0;
#declare foref_y=3.0;
#declare foref_z=0.0;

#declare fb_top=
object{
    basic(0,0,0,1.3,2.5,1.3)
    rotate<0,0,60>
    translate <foref_x,foref_y,foref_z>

}

#declare fb_top_knuckle=
merge{
    object{fb_top}

    translate<-foref_x,-foref_y,-foref_z>
    rotate t_top_r
    translate <foref_x,foref_y,foref_z>

}

```

```
#declare foref_x=-3.0;
#declare foref_y=2.0;
#declare foref_z=0.0;

#declare fb_mid=
merge{
    object{fb_top_knuckle}
    object{
        basic(0,0,0,1.3,2.5,1.3)
        rotate<0,0,60>
        translate <foref_x,foref_y,foref_z>
    }
}
```

```
#declare fb_mid_knuckle=
merge{
    object{fb_mid}
    translate<-foref_x,-foref_y,-foref_z>
    rotate t_mid_r
    translate <foref_x,foref_y,foref_z>
}
```

```
#declare foref_x=-1.3;
#declare foref_y=1.0;
#declare foref_z=0.0;
```

```
#declare fb_down=
merge{
    object{fb_mid_knuckle}
    object{
        basic(0,0,0,1.2,1.2,1.2)
        rotate<0,0,70>
        translate <foref_x,foref_y,foref_z>
    }
}
```

```
merge{
    object{fb_down}
    translate<-foref_x,-foref_y,-foref_z>
    rotate t_down_r
}
```

```
translate <foref_x,foref_y,foref_z>  
pigment{color Pink}
```

```
}
```

## Appendix C Model Template Three Source Code

```
//quadric_2 (super quadric)

#include "colors.inc"
#include "textures.inc"
#include "shapes.inc

##declare camera_location=<0,10,-20>; //look from back
##declare camera_location=<30,6,>; //look from little finger.
#declare camera_location=<0,6,20>; //look from front.
##declare camera_location=<-20,6,0>; //look from thumb finger
##declare camera_location=<0,20,0>; //look from top

#declare camera_lookat=<0,6,0>;
camera {
    location camera_location
    look_at camera_lookat
}

##declare light_location= <0,6,20>;
#declare light_location=camera_location;

light_source {light_location color White}

#macro basic(x0,y0,z0,WX,HY,DZ) //ellipsoid H for y axis, w for x axis, d for z axis

    #local aa = HY*HY*DZ*DZ;
    #local bb = WX * WX * DZ * DZ;
    #local cc = WX * WX * HY * HY;
    #local gg = -(2 * x0 * HY * HY * DZ * DZ);
    #local hh = -(2 * y0 * WX * WX * DZ * DZ);
    #local ii = -(2 * z0 * WX * WX * HY * HY);
    #local jj = (HY * HY * DZ * DZ * x0 * x0) + (WX * WX * DZ * DZ * y0 * y0) + (WX * WX
    * HY * HY * z0 * z0) - (WX * WX * HY * HY * DZ * DZ);

    quadric {<aa,bb,cc >,<0,0,0>,<gg,hh,ii>,jj}
#end

//=====bone=====
#macro bone(x0,y0,z0,RADIUS,HEIGHT) //Connect one spheres with a cylinder.
    //for top sphere

    #local t_y0 = y0 + (HEIGHT/2);
```

```

#local t_aa = RADIUS*RADIUS*RADIUS*RADIUS;
#local t_bb = RADIUS * RADIUS * RADIUS * RADIUS;
#local t_cc = RADIUS * RADIUS * RADIUS * RADIUS;
#local t_gg = -(2 * x0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local t_hh = -(2 * t_y0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local t_ii = -(2 * z0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local t_jj = (RADIUS * RADIUS * RADIUS * RADIUS * x0 * x0) + (RADIUS * RADIUS
* RADIUS * RADIUS * t_y0 * t_y0) + (RADIUS * RADIUS * RADIUS * RADIUS * z0 * z0) -
(RADIUS * RADIUS * RADIUS * RADIUS * RADIUS * RADIUS);

```

```

//for cylinder
#local aaa = RADIUS*RADIUS;
#local ccc = RADIUS*RADIUS;
#local ggg = -(2 * x0 * RADIUS * RADIUS);
#local iii = -(2 * z0 * RADIUS * RADIUS);
#local jjj = (RADIUS * RADIUS * x0 * x0) + (RADIUS * RADIUS * z0 * z0) - (RADIUS *
RADIUS * RADIUS * RADIUS);

```

```

//for plane
#local y1 = y0 + (HEIGHT/2);
#local y2 = y0 - (HEIGHT/2);
#local h4 = -y1-y2;
#local j4 = y1*y2;

```

```

merge{
  //top sphere
  quadric {<t_aa,t_bb,t_cc >,<0,0,0>,<t_gg,t_hh,t_ii>,t_jj}
  intersection{
    //cylinder
    quadric {<aaa,0,ccc>,<0,0,0>,<ggg,0,iii>,jjj}
    //plane
    quadric {<0,1,0>,<0,0,0>,<0,h4,0>,j4}
  }
}

```

```

}
#end

```

```

//=====bone=====
#macro mp(x0,y0,z0,RADIUS,HEIGHT) //shpere as joint

```

```

//for down sphere
#local RADIUSS=RADIUS;

```

```

#local d_y0 = y0 - (HEIGHT/2);
#local d_aa = RADIUS*RADIUS*RADIUS*RADIUS;
#local d_bb = RADIUS * RADIUS * RADIUS * RADIUS;
#local d_cc = RADIUS * RADIUS * RADIUS * RADIUS;
#local d_gg = -(2 * x0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local d_hh = -(2 * d_y0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local d_ii = -(2 * z0 * RADIUS * RADIUS * RADIUS * RADIUS);
#local d_jj = (RADIUS * RADIUS * RADIUS * RADIUS * x0 * x0) + (RADIUS *
RADIUS * RADIUS * RADIUS * d_y0 * d_y0) + (RADIUS * RADIUS * RADIUS *
RADIUS * z0 * z0) - (RADIUS * RADIUS * RADIUS * RADIUS * RADIUS *
RADIUS);

```

```

quadric{<d_aa,d_bb,d_cc>,<0,0,0>,<d_gg,d_hh,d_ii>,d_jj}

```

```

#end

```

```

//=====

```

```

##declare hand_origin=0.0;
##declare hand_origin=0.0;
##declare hand_origin=0.0;
#declare knuckle_0=.5;

```

```

//change these knuckle rotate angle then you can get lots of different gestures.
//the thumb finger still can not move.
//e.g middle, ring and little all top, mid and down knuckle to x axis to 80 you

```

```

//=====

```

```

//index finger
#declare f_x=-1.9;
#declare f_y=5.8;
#declare f_z=0.0;

```

```

#declare f_top_r=index_top_knuckle;
#declare f_mid_r=index_mid_knuckle;
#declare f_down_r=index_down_knuckle;

```

```

#declare f_size_wx=1;
#declare f_size_hy=1.7;
#declare f_size_dz=1;

```

```

#declare knuckle=f_size_wx-1;

#declare finger_top=
merge{

    object{bone(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy)}
}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{bone(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy)}
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{bone(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
}

merge{
    object{finger_down}
    object{mp(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
    pigment{color Pink}
}

```

```

        translate<-f_x,-f_y,-f_z>
        rotate f_down_r
        translate<f_x,f_y,f_z>
    }

//=====
//middle finger
#declare f_x=0.1;
#declare f_y=7.0;
#declare f_z=0.0;

#declare f_top_r=middle_top_knuckle;
#declare f_mid_r=middle_mid_knuckle;
#declare f_down_r=middle_down_knuckle;

#declare f_size_wx=1;
#declare f_size_hy=1.9;
#declare f_size_dz=1;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

    object{bone(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy)}
}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{bone(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy)}
}

```

```

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{bone(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
}

merge{
    object{finger_down}
    object{mp(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
    pigment{color Pink}
    translate<-f_x,-f_y,-f_z>
    rotate f_down_r
    translate<f_x,f_y,f_z>
}

//=====
//Ring finger
#declare f_x=2.1;
#declare f_y=6.3;
#declare f_z=0.0;

#declare f_top_r=ring_top_knuckle;
#declare f_mid_r=ring_mid_knuckle;
#declare f_down_r=ring_down_knuckle;

#declare f_size_wx=1;
#declare f_size_hy=1.8;
#declare f_size_dz=1;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

```

```

        object{bone(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy)}
    }

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{bone(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy)}
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}
    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{bone(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
}

merge{
    object{finger_down}
    object{mp(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
    pigment{color Pink}
    translate<-f_x,-f_y,-f_z>
    rotate f_down_r
    translate<f_x,f_y,f_z>
}

```

```

//=====
//Little finger
#declare f_x=4.0;
#declare f_y=6.0;
#declare f_z=0.0;

#declare f_top_r=little_top_knuckle;
#declare f_mid_r=little_mid_knuckle;
#declare f_down_r=little_down_knuckle;

//
#declare f_size_wx=.8;
#declare f_size_hy=1.5;
#declare f_size_dz=.8;

#declare knuckle=f_size_wx-.1;

#declare finger_top=
merge{

    object{bone(f_x,f_y+3*f_size_hy,f_z,f_size_wx,f_size_hy)}
}

#declare finger_top_Knuckle=
merge{
    object{finger_top}
    translate<-f_x,-(f_y+2*f_size_hy+knuckle),-f_z>
    rotate f_top_r
    translate<f_x,f_y+2*f_size_hy+knuckle,f_z>
}

#declare finger_mid=
merge{
    object{finger_top_Knuckle}
    object{bone(f_x,(f_y+2*f_size_hy),f_z,f_size_wx,f_size_hy)}
}

#declare finger_mid_Knuckle=
merge{
    object{finger_mid}

```

```

    translate<-f_x,-(f_y+f_size_hy+knuckle),-f_z>
    rotate f_mid_r
    translate<f_x,(f_y+f_size_hy+knuckle),f_z>
}

#declare finger_down=
merge{
    object{finger_mid_Knuckle}
    object{bone(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
}

merge{
    object{finger_down}
    object{mp(f_x,f_y+f_size_hy,f_z,f_size_wx,f_size_hy)}
    pigment{color Pink}
    translate<-f_x,-f_y,-f_z>
    rotate f_down_r
    translate<f_x,f_y,f_z>
}

//=====
//plam
#declare f_x=0.9;
#declare f_y=3.5;
#declare f_z=0.0;

merge{
    object{
        basic(f_x,f_y,f_z,4,4,1.3)
        //pigment {color Pink}
    }
}
//=====
#declare f_x=2.7;
#declare f_y=3.6;
#declare f_z=0.0;

object{
    basic(0,0,0,2.3,4.1,1.2)
    rotate<0,0,-10>
    translate <f_x,f_y,f_z>
}

```

```

        //pigment {color Pink}

    }
    //
    #declare f_x=1.3;
    #declare f_y=5.3;
    #declare f_z=0.0;
    object{
        basic(0,0,0,3.75,2.5,1.35)
        rotate<0,0,-5>
        translate <f_x,f_y,f_z>
        //pigment {color Pink}

    }

    #declare f_x=-0;
    #declare f_y=3.8;
    #declare f_z=0.0;
    object{
        basic(0,0,0,2.7,3.9,1.3)
        rotate<0,0,30>
        translate <f_x,f_y,f_z>
        //pigment {color Blue}

    }
}
//=====
//thumb base
#declare f_x=-0.0;
#declare f_y=2.6;
#declare f_z=0.0;

object{
    basic(0,0,0,2.7,3.6,1.32)
    rotate<0,0,40>
    translate <f_x,f_y,f_z>
    //pigment {color Yellow}

}

pigment {color Pink}

}
//=====
//thumb

```

```

##declare t_top_r=thumb_top_knuckle;
##declare t_mid_r=thumb_mid_knuckle;
##declare t_down_r=thumb_down_knuckle;

#declare t_top_r=<0,0,0>;
#declare t_mid_r= <0,0,0>;
#declare t_down_r=<0,0,0>;

#declare foref_x=-5.0;
#declare foref_y=3.0;
#declare foref_z=0.0;

#declare fb_top=
object{
    basic(0,0,0,1.3,2.5,1.3)
    rotate<0,0,60>
    translate <foref_x,foref_y,foref_z>

}

#declare fb_top_knuckle=
merge{
    object{fb_top}

    translate<-foref_x,-foref_y,-foref_z>
    rotate t_top_r
    translate <foref_x,foref_y,foref_z>

}

#declare foref_x=-3.0;
#declare foref_y=2.0;
#declare foref_z=0.0;

#declare fb_mid=
merge{
    object{fb_top_knuckle}
    object{
        basic(0,0,0,1.3,2.5,1.3)
        rotate<0,0,60>
        translate <foref_x,foref_y,foref_z>

    }
}

```

```

}

#declare fb_mid_knuckle=
merge{
  object{fb_mid}
  translate<-foref_x,-foref_y,-foref_z>
  rotate t_mid_r
  translate <foref_x,foref_y,foref_z>

}

#declare foref_x=-1.3;
#declare foref_y=1.0;
#declare foref_z=0.0;

#declare fb_down=
merge{
  object{fb_mid_knuckle}
  object{
    basic(0,0,0,1.2,1.2,1.2)
    rotate<0,0,70>
    translate <foref_x,foref_y,foref_z>
  }
}

merge{
  object{fb_down}
  translate<-foref_x,-foref_y,-foref_z>
  rotate t_down_r
  translate <foref_x,foref_y,foref_z>
  pigment{color Pink}
}

```

## Appendix D C++ Program Source Code

```
#include <iostream> // I/O
#include <fstream> // file I/O
#include <iomanip> // format manipulation
#include <string>
#include <complex>
#include <vector>

using namespace std;

string filename_i;
string aslname_i;
string filename_o;

int main(int argc, char** argv)
{
    char seek[100];
    char asl[100];
    char str[400];
    char sign;

    /* if(argc==4){
        asl=argv[1];
        gesture=argv[2];
        filename_o=atoi(argv[3]);

    }
    else {
        printf(" Error input format!!\n");
        printf("e.g. Hand asl y output.pov \n");
        exit(0);
    }*/

    cout<<"3 template: sphere.pov quadric_1.pov quadric_2.pov"<<endl;
    cout << "please input filename(*.pov): ";
    cin >> filename_i;
    filename_i="quadric_1.pov";
    cout<<"Using template :"<<filename_i<<endl<<endl;

    aslname_i="asl.txt";
    cout<<"Gesture file:"<<aslname_i<<endl<<endl;
```

```

cout << "Which gesture(i.e. y or l or 3 or 5):  ";
cin >> sign;

filename_o="ASL.pov";

ifstream inasl;
inasl.open (aslname_i.c_str());

ofstream outfile(filename_o.c_str());

if (!inasl){
    cout << "Error: Could not open file "<<aslname_i<<" !"<<endl;
    //cin.get();
    exit(EXIT_FAILURE);
}

int flag=0;
while(inasl && flag==0)
{
    inasl.getline(seek,100);
    //cout<<seek[0]<<endl;
    if (seek[0]==sign){
        flag=1;
        for (int a=0; a<=15; a++){
            inasl.getline(asl,100);
            outfile <<asl<<endl;
        }
    }
}
inasl.close();

if (flag==0){
    cout << "Error: Can't this gesture "<<sign<<" !"<<endl;
    //cin.get();
    exit(EXIT_FAILURE);
}

ifstream infile;
infile.open (filename_i.c_str());
if (!infile){
    cout << "Error: Could not open file "<<filename_i<<" !"<<endl;
    //cin.get();
    exit(EXIT_FAILURE);
}

```

```
while(infile)
{
    infile.getline(str,400);
    outfile <<str<<endl;
}

infile.close();
outfile.close();
cout<<endl<<"The output POV-Ray file is: "<<filename_o<<endl;
}
```

## Appendix E **asl.txt File (Gesture Information)**

1

```
#declare index_top_knuckle=<0,0,0>;//1
#declare index_mid_knuckle=<0,0,0>;
#declare index_down_knuckle=<0,0,0>;

#declare middle_top_knuckle=<70,0,0>;
#declare middle_mid_knuckle=<70,0,0>;
#declare middle_down_knuckle=<70,0,0>;

#declare ring_top_knuckle=<70,0,0>;
#declare ring_mid_knuckle=<70,0,0>;
#declare ring_down_knuckle=<70,0,0>;

#declare little_top_knuckle=<70,0,0>;
#declare little_mid_knuckle=<70,0,0>;
#declare little_down_knuckle=<70,0,0>;
```

y

```
#declare index_top_knuckle=<70,0,0>;//y
#declare index_mid_knuckle=<70,0,0>;
#declare index_down_knuckle=<70,0,0>;

#declare middle_top_knuckle=<70,0,0>;
#declare middle_mid_knuckle=<70,0,0>;
#declare middle_down_knuckle=<70,0,0>;

#declare ring_top_knuckle=<70,0,0>;
#declare ring_mid_knuckle=<70,0,0>;
#declare ring_down_knuckle=<70,0,0>;

#declare little_top_knuckle=<0,0,0>;
#declare little_mid_knuckle=<0,0,0>;
#declare little_down_knuckle=<10,0,-20>;
```

3

```
#declare index_top_knuckle=<0,0,0>;//3
#declare index_mid_knuckle=<0,0,0>;
#declare index_down_knuckle=<0,0,10>;

#declare middle_top_knuckle=<0,0,0>;
#declare middle_mid_knuckle=<0,0,0>;
#declare middle_down_knuckle=<10,0,-5>;
```

```
#declare ring_top_knuckle=<70,0,0>;  
#declare ring_mid_knuckle=<70,0,0>;  
#declare ring_down_knuckle=<70,0,-5>;
```

```
#declare little_top_knuckle=<70,0,0>;  
#declare little_mid_knuckle=<70,0,0>;  
#declare little_down_knuckle=<70,0,-10>;
```

5

```
#declare index_top_knuckle=<0,0,0>;//5  
#declare index_mid_knuckle=<0,0,0>;  
#declare index_down_knuckle=<0,0,10>;
```

```
#declare middle_top_knuckle=<0,0,0>;  
#declare middle_mid_knuckle=<0,0,0>;  
#declare middle_down_knuckle=<0,0,0>;
```

```
#declare ring_top_knuckle=<0,0,0>;  
#declare ring_mid_knuckle=<0,0,0>;  
#declare ring_down_knuckle=<0,0,-10>;
```

```
#declare little_top_knuckle=<0,0,0>;  
#declare little_mid_knuckle=<0,0,0>;  
#declare little_down_knuckle=<0,0,-20>;
```