PROGRAMACION: construyendo un mundo virtual...

Juan Carlos Parra Márquez

5.1. VRML

1. Descripción General.

Es de importancia analizar primero el lenguaje VRML por tratarse de un estándar para ambientes virtuales en Internet. De hecho, la mayoría de los programas de RV han ido incorporando la exportación de sus diseños en este formato. Los diseños en VRML se basan en la interpretación y despliegue mediante programas denominados "Browser", los cuales son visualizadores que interpretan el código y a partir de esto presentan el ambiente renderizando las imágenes correspondientes.

VRML se difunde a partir de la conferencia anual sobre World-Wide-Web realizada en marzo de 1994. Los autores, Tim Bernes Lee y David Roget, presentaron una ponencia titulada "Los Lenguajes de Marcación de Realidad Virtual y el Word Wide Web" y a partir de esta presentación los asistentes a la conferencia se comprometieron a delinear los requerimientos básicos para generar un producto para diseño 3D, que fuese equivalente al estándar HTML. Una propuesta inicial, VRML 1.0, se presentó en la asamblea de octubre del mismo año. Esta versión fue oficialmente lanzada en abril de 1995. La nueva versión posterior, 2.0, crea ambientes virtuales multiusuarios e interactivos donde los navegantes pueden ser participantes y relacionarse entre sí.

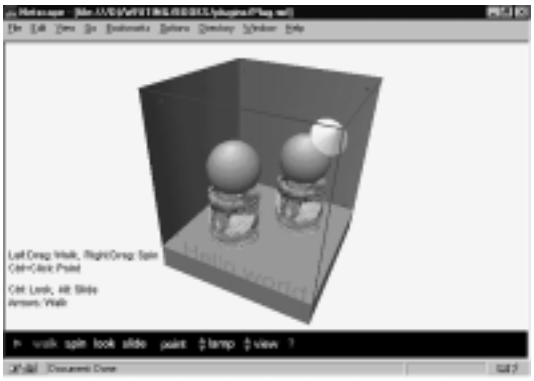


Fig. 5.1 - Diseño de VRML visualizado mediante browser.

VRML se puede definir como un lenguaje cuyo texto básico está en código ASCII y sus ambientes generalmente se reconocen por la extensión WRL. El diseño de formas se basa en los conceptos usuales de la computación gráfica. Muchos de los productos que generan mundos virtuales para Web se basan en VRML y éstos cuentan con generadores de formas, en los cuales el usuario no trabaja directamente en VRML sino que cuenta con interfaces que le permiten diseñar directamente a partir de objetos preestablecidos.

Cada escena tiene un punto de vista, el cual es llamado cámara. El usuario puede ver la escena a través de los ojos de la cámara. También es posible predefinir otros puntos de vista por el creador del mundo virtual. Muchos de los conceptos definidos para los productos de RV son válidos para VRML. De los Browser depende el despliegue de un ambiente virtual en Internet. Algunos no pueden trabajar la aplicación directamente, es decir, necesitan integrarse en otros navegadores de Internet. Estos envían sus requerimientos al browser Web y este último envía los mismos a la red (servidor). Generalmente, el Net Browser es Netscape o Explorer. Esto se puede entender gráficamente mediante el siguiente esquema

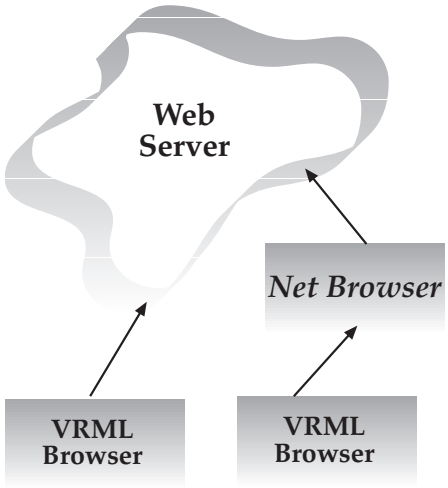


Fig. 5.2.

Un mundo VRML puede estar distribuido, es decir, residir en múltiples puntos de la red en distintos lugares geográficos, de la misma manera que una página HTML puede estar compuesta de texto en un lugar y de imágenes en otro. Un mundo VRML puede especificar que los escenarios estén en un servidor, mientras que en otro están los objetos para dicha escena.VRML tiene similitud a otros lenguajes de programación tales como BASIC o C, pero ha sido diseñado específicamente para ser un lenguaje de computación gráfica. En realidad es un lenguaje de descripción de escenas y no un lenguaje de programación propiamente tal. Otros lenguajes, como C, son compilados en programas ejecutables mientras que VRML es interpretado y a partir de esto es desplegado en pantalla. El escenario virtual es una descripción estática, es decir, nada cambia el archivo VRML una vez en ejecución, lo que sí se puede cambiar son los puntos de vistas, pero no las escenas propiamente tales.

2. Nodos.

Un documento VRML consiste de un listado de objetos conocidos como Nodos (*nodes*). Están constituidos por una estructura jerárquica, y pueden estar conformados, a su vez, por otros nodos. Esta lista completa de nodos es conocida como escena gráfica. En este sentido, cada documento VRML es una escena gráfica.

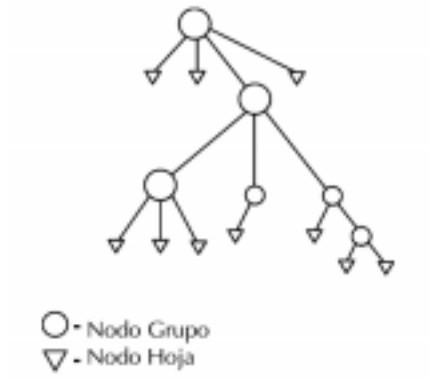


Fig. 5.3 - Jerarquización de formas.

Primero existen nodos "tipo", los cuales determinan las características en la escena gráfica. Ejemplos de nodos tipo son *Sphere* (esfera), *cube* (cubo), *WWWInline* (conexión en línea) y *Separator* (separador). Los nodos pueden tener más campos. Campos son lugares para almacenar información específica para el nodo. Por ejemplo, el nodo *Sphere* tiene un campo llamado *Radius* (radio) con el cual se le suministra un valor para el radio. Algunos nodos, conocidos como *Group nodes* (nodos de grupo), pueden tener otros nodos dentro de ellos. Los nodos de grupo son el equivalente en VRML, a una caja de objetos. Un objeto en la caja puede ser tratado como una unidad y puede ser pintado, cambiar su tamaño o deformado con una misma operación. Los nodos de grupo son un concepto muy importante

en VRML, ellos dan el poder para manipular muchos objetos con una simple operación. El nodo de grupo más utilizado es el *Separator*, que actúa como una caja genérica. En suma, los cambios pueden ser realizados en los ítemes de la caja sin afectar a los que estén fuera de ella. Por ejemplo, en el nodo Material se pueden especificar las cualidades de una superficie, tales como: color, luz, reflexibilidad, y otras.

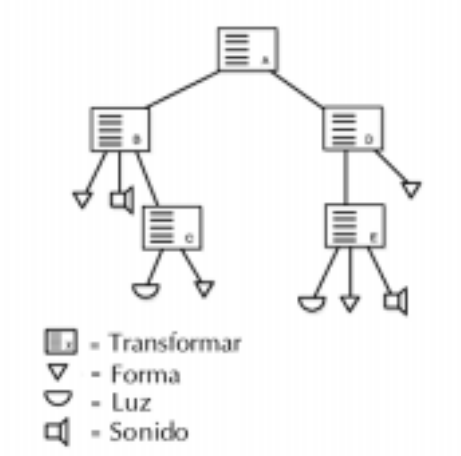
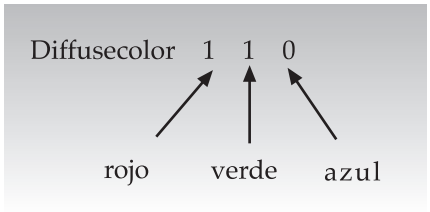


Fig. 5.4 - Estructura de un nodo.

Poniendo un nodo *Sphere* dentro de un grupo *Separator*, precedido por un nodo *Material*, es posible asociar "color" a la esfera a partir de *Material*. Si quisiéramos una esfera amarilla, debemos poner la *Sphere* dentro de un nodo de grupo y luego ponemos un nodo *Material* antes de ella. El nodo *Material* tiene algunos campos con los cuales se pueden crear características visibles como color (*color*) y brillo (*shiness*). El campo en el nodo Material usa un set básico de colores y éste es llamado *diffusecolor*. Este campo puede ser especificado con tres valores RGB (Rojo, verde, azul), en números entre 0 y 1. Por ejemplo, si deseamos el color amarillo (verde + rojo) se definiría:



Otro nodo es *Transform*, este cambia la posición, orientación y tamaño dentro de algunos nodos que lo sigan dentro de un grupo. Para mover la posición de secuencia de nodos, el nodo *transform* usa el campo traslation, el cual provee los valores de x, y, z. Definiendo algunos de estos valores en 0 no produce cambio. En general, tenemos en la actualidad dos versiones de VRML, la 1.0 y la 2.0, las cuales trataremos básicamente mediante la definición de algunos de sus nodos. Esto se adiciona a las definiciones anteriormente expuestas.

3. Sintaxis básica de VRML.

VRML es, en esencia, un archivo en formato ASCII y tiene una estructura similar a las páginas HTML. Para comenzar, se pueden incorporar líneas de comentarios mediante el símbolo #. La primera línea de todo archivo VRML es la llamada cabeza de definición, la cual define la identificación del archivo, detallando la versión de VRML que estemos utilizando.

```
# VRML V1.0 ascii
```

para VRML 2.0 esto es similar.

```
#VRML V2.0 utf8
```

o bien,

```
#VRML Draft #n V2.0 utf8
```

donde n es el número de la escena gráfica asignada por el autor.

A continuación vienen contenidos, los nodos de la aplicación. Por ejemplo,

DEF nombre tiponodo {campos hijos }

<DEF nombre> es opcional. Tiponodo corresponde a la identificación del tipo de nodo al que corresponde (ejemplo: *Sphere*, *Separator*, etc.). Campos es una lista de campos formada por el par nombre-valor que especifica los parámetros de los nodos, e hijos son nodos adicionales.

En general un nodo se describe bajo la forma:

NodoTipo { campos }

Un pequeño ejemplo de VRML 1.0 sería:

```
#VRML V1.0 ascii

Group {
    PointLight { position 0 15 0 }
        # una fuente de luz a 15 mts.
    Material { diffuseColor 0 0 1 } # color azul brillante
    Sphere { radius 2.0 } # una esfera de 2 mts.
}
```

Este archivo contiene 4 nodos. En particular, el nodo *PointLight* posee un solo campo, pero éste contiene tres valores que corresponden a la posición en los ejes x, y, z. Finalmente, el nodo *Group* no posee campos, pero contiene 3 hijos, los nodos: *PointLight, Material y Sphere.*

El detalle de algunos nodos de mayor uso es el siguiente:

DEF y USE

El estamento DEF lo habilita para asignar un nombre a un nodo a crear. USE se utiliza para duplicar una shape tantas veces como se quiera. Un ejemplo sería:

```
# VRML V1.0 ascii
Group { # nodo grupo
        DEF cameras Switch { # multiples cameras
            whichChild 0
            DEF First_camera PerspectiveCamera {
                position 1.5 1.5 7
            }

            DEF Second_camera PerspectiveCamera {
                position 1.5 7 1.5
                orientation 1 0 0 -1.5873
            }

            DEF Third_camera PerspectiveCamera {
                position 7 1.5 1.5
                orientation 0 1 0 1.5873
            }
        }
```

SHAPES

VRML posee 8 diferentes tipos de primitivas que pueden ser usadas para crear todo objeto visible en el mundo VRML. Algunas de esta shapes primitivas son los nodos: *Sphere* (esfera), *Cube* (cube), *Cone* (cono) y *Cylinder* (cilindro).

```
Shape {
    appearance Appearance {
        material Material {
            diffuseColor 0 0.5 0.5 transparency 0.8
        }
        } geometry Cylinder { }
        }
```

Nodo ASCII TEXT

El nodo *AsciiText* permite crear objetos texto con texto. El nodo *Pointset* especifica una "constelación" de puntos en el espacio 3D y *IndexedLineSet* permite crear objetos que consisten de segmentos de líneas. Ejemplo:

```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { diffuseColor 0 0 0 }
    }
    geometry Text {
        string [ "Hola!" "¿Cómo estas?" ]
        fontStyle FontStyle {
            family [ "Serenity", "SERIF" ]
        }
        length [ 4 6 ]
    }
}
NavigationInfo { type "EXAMINE" }
Viewpoint { description "start" position 4 0 10 }
```

Nodos POINTSET, INDEXEDLINESET e INDEXEDFACESET

Tanto los nodos *PoinSet*, *IndexedLineSet* y *IndexedFaceSet* usan una lista de vértices coordenados que el nodo *Coordinate3* almacena. Además VRML introduce 2 nuevos nodos *geometry*: *ElevationGrid* para crear mallas (terrenos) y *Extrusion* para crear superficies de prolongación o de revolución.

LIGHTS

Existen 3 tipos de luces: *DirectionalLight*, *PointLight* y *SpotLight*. El primero, ilumina mediante líneas paralelas en una dirección específica. *PointLight* es similar a la luz de una ampolleta, su luz se irradia en todas direcciones. *SpotLight* posee forma de cono, el cual determina cuales objetos alumbrará. Un ejemplo es:

```
DirectionalLight {
    on TRUE
    intensity 1
    direction 0 0 -1
    color 1 1 0
}
```

Nodos GROUP y SEPARATOR

VRML 1.0 posee varios nodos de grupo que pueden contener otros nodos como hijos. Se puede usar el nodo *Group*, pero el más usado es *Separator*. Este último limita el rango de efectos de *Material*, *Texture2*, *Transform* y otros nodos. Ejemplo:

```
Group {
    children [ Shape { geometry { Sphere { } }, Shape {
geometry Cone { } } ]
    }
```

Nodo TRANSFORM

Se utiliza para especificar la localización, orientación y tamaño de objetos en la aplicación virtual. Una forma típica de utilización sería:

```
Transform {
        scaleFactor 21 34 45
        translation 12 14 27
        rotation 0 1 0 1.5708
    }
```

en general, su formato clásico es:

```
Transform {
    translation X Y Z
    rotation AX AY AZ ANGLE
    scale SX SY SZ
    scaleOrientation OX OY OZ ANGLE
    center X Y Z children [ ... ]
    }
```

en donde todos los parámetros tienen relación con sus coordenadas en los ejes x, y, z.

Nodos GEOMETRY

Para VRML 2.0 se tienen 10 nodos de geometría. Estos para VRML 1.0 son principalmente: *Sphere, Cylinder y Cone*. El nodo *Text* es uno de ellos y para este tipo de nodos se puede usar una variedad de tamaños y estilos. Un ejemplo sería:

```
# VRML V2.0 utf8

Shape {
    appearance Appearance {
        material Material { diffuseColor 0 0 0 }
    }

  geometry Text {
            string [ "Hola!" "¿Cómo estás?" ]
            fontStyle FontStyle {
            family [ "Serenity", "SERIF" ]
        }
        length [ 4 6 ]
    }
}
NavigationInfo { type "EXAMINE" }
Viewpoint { description "start" position 4 0 10 }
```

Nodo APPEARANCE

Este posee tres campos:

```
Appearance {
    material <Nodo Material>
    texture <Nodo Textura>
    textureTransform <Nodo Transformación>
    }
```

es evidente la interpretación de este nodo. Sólo explicaremos el campo *textureTransform* el cual permite estirar, trasladar y girar texturas sobre las caras de un objeto. Se puede especificar las texturas con *ImageTexture* (permite 'cargar' una imagen desde archivos JPEG o PNG), con *MovieTexture* se puede mover una textura JPEG a una cara (*face*) de un objeto o utilizar *PixelTexture* con el cual se puede generar una textura.

Nodo SOUND

El sonido se ha transformado en un importante aporte a escenas gráficas en VRML y para ello se tiene el nodo *Sound* el cual es acompañado del nodo *AutoClip*. Este último carga un archivo de sonido desde Internet y puede tocar el sonido en forma de loop infinito o puede tocar en cierta ocasión. El nodo Sound posee los controles de localización, orientación, intensidad y rango de efectos. Un ejemplo es:

```
Sound {
        location 15.5 23 0.2
        intensity 0.75
        source AudioClip {
            description "campana"
            loop TRUE
            url "http://somewhere.com/sounds/bell.wav"
        }
    }
```

otro ejemplo sería:

```
Sound {
    source AudioClip {
        url [ "Sounds/wind.wav" ]
        loop TRUE
    }
    spatialize FALSE
    intensity .25
    minFront 20
    maxFront 20
    minBack 20
    maxBack 20
}

Sound {
    source AudioClip {
        url [ "doh.wav" ]
        loop TRUE
    }
    spatialize TRUE
    intensity 1
    minFront 0
    maxFront 8
    minBack 0
    maxBack 8
}
```

VIEWPOINTS

El concepto de cámaras ha cambiado un poco; en realidad, para VRML 2.0 son llamados puntos de vista (*Viewpoints*). Estos almacenan una localización, orientación y campos de vista y es similar a la versión del nodo *PerspectiveCamera* de VRML 1.0, pero se pueden ubicar múltiples puntos de vistas en una escena. Ejemplo:

```
Viewpoint {
    description "Izquierda"
    position -7 0 7
    orientation 0 1 0 -.79
}

Viewpoint {
    description "Frontal"
    position 0 0 9
}

Viewpoint {
    description "Derecha"
    position 7 0 7
    orientation 0 1 0 .79
}
```

Nodo SWITCH

Este nodo permite renderizar uno de sus hijos o ninguno de ellos. Posee el campo *whichChoice* que especifica cuál hijo renderizará (comenzando desde cero). Si *whichChoice* es menor que cero, ninguno de los hijos es procesado. En el ejemplo siguiente, se selecciona uno de ellos de las diferentes *shapes* que contienen dependiendo del valor de *whichChoice*:

```
Switch {
        whichChoice 2
        choices [
            Shape { geometry Sphere { } }
            Shape { geometry Box { } }
            Shape { geometry Cone { } }
        ]
    }
```

El campo *whichChoice* tiene como valor 2, así la shape inicial es el cono (porque los hijos son contados desde cero). El nodo Switch es usado para activar ciertas partes de una escena cuando ésta no es visible y para crear una simple animación que consiste de múltiples versiones de un mismo objeto.

Nodos ANCHOR e INLINE

Estos son la versión de VRML 2.0 para *WWWAnchor* y *WWWInline* de VRML 1.0. Sirven para cargar archivos VRML y sumarlos a otro mundo. Su forma es:

```
Inline { url "archivo.wrl" }
```

Un ejemplo más completo es:

```
# VRML V2.0 utf8
  Group { children [
    Transform {
      translation -5 0 0
        children Anchor {
          url "http://www.barbie.web/~barbie/dollhouse.wrl"
            description "llamar la página Web"
              children Shape {
                appearance DEF A1 Appearance {
                  material Material {
                    diffuseColor 1 1 1  ambientIntensity 0.33
                      specularColor 1 1 1  shininess 0.5
                    }
                  }
                geometry Box {}
              }
          }
        }
```

Nodo LOD

Este, básicamente, es similar en las dos versiones de VRML. Permite especificar múltiples niveles de detalle para un objeto. La velocidad de recorrido de una escena depende de la complejidad de la misma, es decir, de la cantidad de caras visibles que existan. El nodo LOD es una poderosa herramienta para mantener esta dinámica óptima de manipulación de objetos. LOD habilita al navegador de VRML, seleccionar una versión del objeto con menor detalle para reducir el procesado de la escena. Un ejemplo sería:

```
LOD {
        range [ 10, 27 ]
        level [
            Inline { url "redondo.wrl" }
            Shape { geometry Sphere { } }
            Shape { geometry Box { } }
        ]
    }
```

Detección de Colisión

El nodo Collision previene al usuario al pasar a través de alguno de los hijos de los objetos.

Nodo BACKGROUND

Este nodo permite dos acciones: crear un cielo y fondo, o crear una panorámica a la distancia, por ejemplo, panorámica de montañas y estrellas. Este nodo le da el rango de colores al cielo, al fondo ángulos de inclinación y otros detalles. Un ejemplo es:

```
# VRML V2.0 utf8
    Transform { children [
        DEF B1 Background { # cielo gris
        skyColor [ 0 0 0, 1.0 1.0 1.0 ]
        skyAngle 1.6
        groundColor [ 1 1 1, 0.8 0.8 0.8, 0.2 0.2 0.2 ]
        groundAngle [ 1.2, 1.57 ]
        }
    DEF B2 Background { # noche
        backUrl "Bg.gif"
        leftUrl "Bg.gif"
        bottomUrl "Bg.gif"
        frontUrl "Bg.gif"
        rightUrl "Bg.gif"
        topUrl "Bg.gif"
        }
```

Nodos PROXIMITYSENSOR y VISIBILITYSENSOR

El nodo *ProximitySensor* indica en la región del espacio que permite detectar cuando un usuario ha ingresado, ha salido o se mueve alrededor del interior de ella. Este nodo reporta la localización y orientación del usuario dentro de la región. El nodo *Script* puede utilizar esta información para un variado número de propósitos. El nodo *VisibilitySensor* es algo similar, pero reporta cuando la región es visible o invisible.

Nodo TIMESENSOR

Este nodo no detecta la interacción con el usuario pero envía mensajes a intervalos regulares. Puede ser utilizado para generar animaciones.

Nodo SCRIPT

Con este nodo se puede escribir un pequeño programa e interfaces en el mundo VRML. Con este nodo se puede incluir la interacción con datos especiales o dispositivos y comunicación en Internet.

PROTOTYPES

Uno de los serios inconvenientes de VRML 1.0 era la nula relación con el concepto vigente, actualmente, de diseño orientado al objeto *(Object Oriented Programming)*. El OOP es una poderosa técnica de programación, utilizada en muchos proyectos de esta índole. Uno de los conceptos más válidos de OOP es el encapsulamiento de código y datos en un simple objeto. La solución a estos problemas es el uso de *prototypes*. Un prototipo es una colección de nodos, con *Route* puede ser usado para definir un nuevo nodo. El siguiente ejemplo define un nuevo tipo de nodo (automóvil):

```
PROTO Automobile [
    field SFColor body_color 0 0 1
    field SFString bumper_style "modern"
    field SFBool tinted_windshield FALSE
    eventIn SFTime engine_starttime
    ]
```

se puede usar "el nodo creado" de la siguiente manera:

```
    Automobile {
    body_color 1 0 0
    tinted_windshield TRUE
    }
```

otro ejemplo sería:

```
# VRML V2.0 utf8

PROTO MyCyl [
        field SFColor color 1 1 1
        field SFVec3f position 0 0 0
        field SFFloat height 1
        field SFFloat radius 1
        ]
{
    Transform {
        translation IS position
            children [
                Shape {
                    appearance Appearance {
                        material Material {
                            diffuseColor IS color
                        }
                    }
                    geometry Cylinder {
                        height IS height
                        radius IS radius
                    }
                }
                ]
            }
        }
```

```
    MyCyl {
        position -6 0 0
        color 1 0 0
        height .5
        radius .25
    }

    MyCyl {
        position -2 0 0
        color 0 1 0
        height .5
        radius 2
    }

MyCyl {
        position 2 0 0
        color 0 0 1
        height 1
        radius 1
    }

MyCyl {
        position 6 0 0
        color 1 1 0
        height 1.5
        radius .5
    }

NavigationInfo { type "EXAMINE" }
Viewpoint { position 0 0 10 }
Background { skyColor 1 1 1 }
```

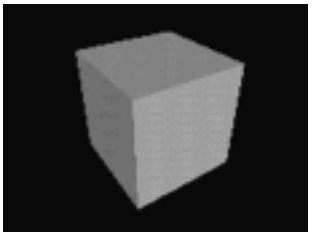
ROUTE

Un route es una conexión entre un campo de salida de un nodo y un campo de entrada de otro. El siguiente ejemplo muestra el aumento rápido en la intensidad de luz al tocar el interruptor:

```
DEF Lightswitch TouchSensor {  }
    DEF Lamp SpotLight { intensity 0.0 }
    DEF Interval TimeSensor { cycleInterval 5 } # segundas
    ROUTE Lightswitch.touchTime TO Interval.startTime
    ROUTE Interval.fraction TO Lamp.intensity
```

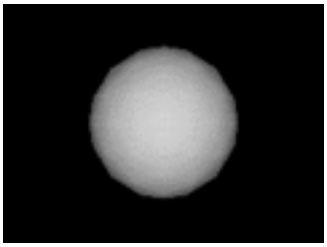
4. Algunos ejemplos de VRML.

Fig. 5.5 Cubo.



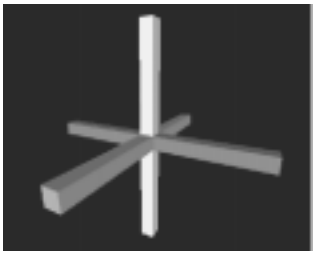
```
# VRML V2.0 utf8
        Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0.8 0.8 0.8
                }
            }
            geometry Box {
                size 2.0 2.0 2.0
            }
        }
```

Fig. 5.6 - Esfera



```
#VRML V2.0 utf8
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.8 0.8 0.8
            }
        }
        geometry Sphere {
            radius 1.0
        }
    }
```

Fig. 5.7 - Cruceta



```
#VRML V2.0 utf8
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.0 0.0 1.0
            }
        }
        geometry Box {
            size 8.0 0.5 0.5
        }
    }
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.0 0.5 1.0
            }
        }
        geometry Box {
            size 0.5 8.0 0.5
        }
    }
Shape {
    appearance Appearance {
        material Material {
            diffuseColor 0.5 0.0 1.0
        }
    }
    geometry Box {
        size 0.5 0.5 8.0
    }
}
```

5.2. V - REALM BUILDER

1. DESCRIPCIÓN DE V-REALM BUILDER.

V-Realm Builder es un poderoso programa que permite la creación de objetos 3D y mundos para ser vistos mediante *V-Realm Browser* o algún otro navegador compatible con VRML. Existe otro producto adicional: *V-Realm Browser Multiuser*, que permite a los usuarios ver e interactuar con otros usuarios en un ambiente 3D. Los usuarios están habilitados para comunicarse con los demás mediante Chat, con audio o texto, así como también pueden intercambiar audio, texto, video, gráficas y archivos binarios mientras estén dentro del mundo 3D. *V-Realm Builder* es una herramienta específica, para desarrollar con facilidad un mundo en VRML. Estos mundos pueden ser generados usando nodos (módulos pre-programados) construidos en el estándar VRML. Los nodos habilitados pueden ser usados en un mundo ya existente o en uno nuevo. En efecto, para que los diseñadores puedan construir un mundo exitoso, primero deben entender los conceptos básicos de una escena gráfica. Una escena es el marco que contiene todos los nodos juntos. La escena gráfica permite a los nodos contener, dentro de él, órdenes y relaciones entre ellos, los cuales, a su turno, determinan cómo renderizar los objetos en la escena que se desplegará. Los objetos son definidos por nodos que describen sus características. Estos objetos poseen forma (*Shape*), tamaño, color, textura y otras características. Es importante recordar que cuando construimos un mundo virtual 3D existe el concepto de objetos superpuestos. No puede poner un sofá en el mismo lugar físico que una silla, es decir, no pueden ocupar el mismo espacio. Sin embargo, en un mundo virtual, cuando se ponen dos objetos en, exactamente, las mismas coordenadas, ellos sólo combinan sus características. Esto permite crear objetos complejos superponiendo muchos objetos de formas más simples.

2. DESCRIPCIÓN DE NODOS.

Los nodos pueden ser accesados mediante menús o mediante iconos. En este texto se hará descripción de su acceso mediante los iconos.

a) Herramientas básicas.

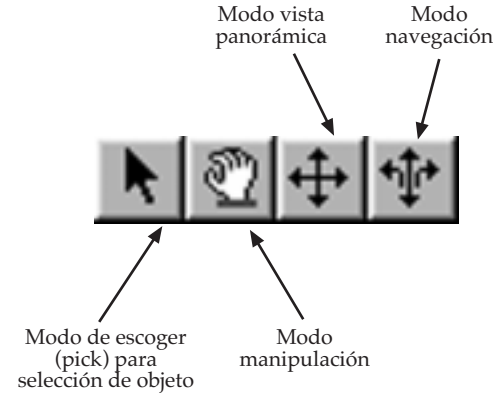


Fig. 5.8

b) Herramientas para manipulación

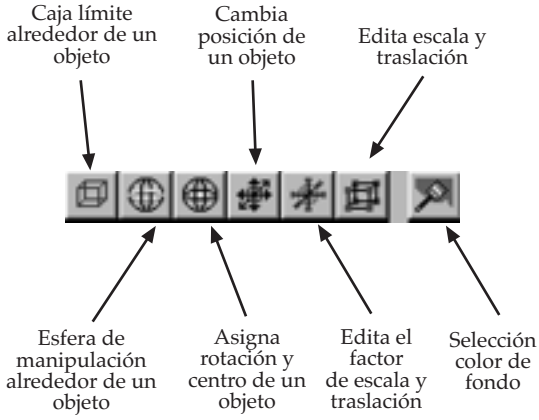


Fig. 5.9

c) Herramientas de Visualización.

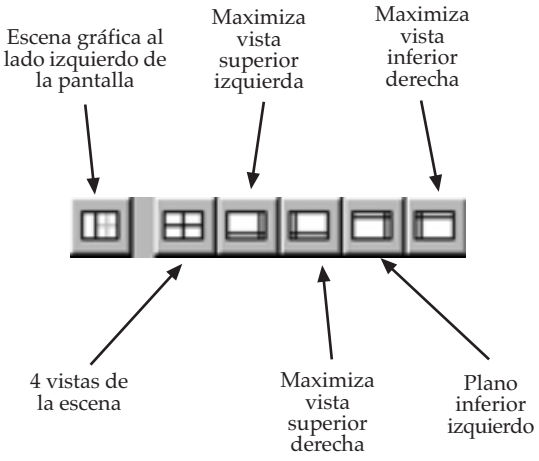
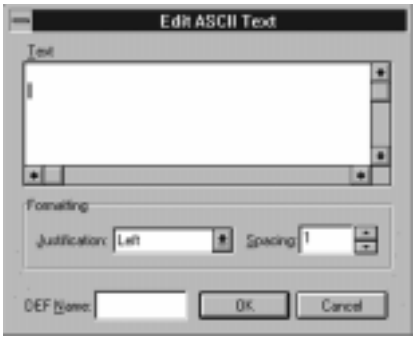


Fig. 5.10

3. LISTADO DE ALGUNOS NODOS MÁS UTILIZADOS.

1. Texto ASCII

Permite ingresar un texto para etiquetar un objeto en particular o dar una dirección para una visualización. El nodo de información de texto es editado a través de la caja de diálogo o Editor de Texto ASCII, que tiene el siguiente aspecto:



2. Cono

Este icono define el nodo de cono y éste queda alineado con el eje x. Este cono está centrado en (0, 0, 0) y tiene un tamaño de -1 a 1 (radio 1) en las tres direcciones como base y altura de 2. El cono está formado por los lados y la base. La información del nodo de cono es indicada a través del Editor de Diálogo.

3. Coordenadas

Define las coordenadas 3D para ser usadas para control de puntos para las formas (Shapes). Estas coordenadas son especificadas como un vector. Este nodo no produce un efecto visible durante la renderización. La información del nodo *Coordinate3* es indicada mediante el *Editor Dialog Box.*

4. Cubo

Este nodo representa un cubo alineado con los ejes. Está centrado en (0, 0, 0) y posee 2 unidades de dimensión por lado, desde -1 a 1. La información del nodo es editable mediante el *Edit Cube Dialog Box.*

5. Cilindro

Representa un cilindro centrado en (0, 0, 0) de tamaño 2 en los tres ejes (-1 a 1). La información del nodo de cilindro es editable mediante el *Edit Cylinder Dialog Box*.

6. Luz Direccionable

Este nodo define una fuente luz dirigida que ilumina con rayos paralelos en forma vectorial 3D. La información puede ser editada mediante el *Edit Directional Light Dialog Box.*

7. Estilo de fondo

Define el tipo de fondo para todos los textos ASCII. La información es editada a través del *Edit Font Style Dialog Box.*

8. Grupo

Es un nodo que contiene una lista de órdenes de nodos hijos. No afecta la escena. La información es editada mediante el *Edit Group Dialog Box.*

9. Información

Es usado para incorporar información a la escena, como autores o algún mensaje específico de utilización. La información es ingresada mediante el *Edit Info Dialog Box.*

10. LOD (Nivel de Detalle)

Es un nodo de grupo que permite a las aplicaciones cambiar automáticamente entre varias representaciones de objetos. Los hijos de este nodo constituyen el mismo objeto a diferentes niveles de detalle, desde el más sofisticado al más simple. La información es ingresada a partir del *Edit LOD Dialog Box.*

11. Material

Define propiedades comunes de las superficies de todas las formas. Para asociar materiales a una forma, se usa el nodo *Material Binding*. Un material se define como un valor de color o transparencia. La información es editada a través del *Edit Material Dialog Box.*

12. Material Binding

Especifica cómo los materiales son asociados a una forma. La información es editada mediante el *Edit Material Binding Dialog Box.*

13. Matriz de Transformación

Define una transformación geométrica 3D con una matriz de 4x4. La información se edita mediante el *Edit Matrix Transform Dialog Box.*

14. Normal

Define vectores 3D perpendiculares a los planos para ser usados como base de vértices de nodos de forma (set de facetas indexadas, set de líneas indexadas, set de puntos, etc.). La información es editada mediante el *Edit Normal Dialog Box.*

15. Normal Binding

Especifica cómo las normales son atadas a las formas que están en la escena gráfica. La información es ingresada en *Edit Normal Binding Dialog Box.*

16. Cámara ortogonal

Define una proyección paralela desde un punto de vista. El volumen que representa una cámara ortogonal es un paralelógramo rectangular (caja). Inicialmente la cámara está fijada en (0, 0, 1) y mira hacia el lado negativo del eje Z. La información del nodo es editada a través del *Edit Orthographic Camera Dialog Box.*

17. Cámara de perspectiva.

Define una proyección de perspectiva desde un punto de vista. El volumen que representa una cámara de perspectiva es una pirámide. Es localizada en (0, 0, 1) y mira hacia el lado negativo del eje Z. El campo *Height Angle* define el ángulo vertical de la vista. La información es editada mediante *Edit Perspective Camera Dialog Box.*

18. Punto de luz

Define el origen de un punto de luz en una localización 3D. Esta luz ilumina de igual manera en todas direcciones. La información es ingresada mediante el *Edit Point Light Dialog Box.*

19. Rotación

Define una rotación 3D a través de algún eje a partir del origen. La información del nodo de rotación es editada mediante *Edit Rotation Dialog Box.*

20. Escalar

Define un escalamiento (cambio de tamaño) 3D referida al origen. Si todos los componentes del vector de escalamiento son diferentes, se produce un tamaño que no es uniforme. La información es editada mediante el *Edit Scale Dialog Box.*

21. Separador

Es un nodo de grupo que lleva a cabo un almacenamiento de su estado antes de atravesar los hijos, y una recuperación al atravesarlos. En otras palabras, esto aisla a los hijos del resto de la escena. Un separador puede incluir luces, cámaras, coordenadas, normales y otras propiedades. La información del nodo es editada a través del *Edit Separator Dialog Box.*

22. Shape indirecta

Nodo que indica un conjunto de índices de facetas de un sólido. Contiene los vértices o las facetas convexas. Esto permite a VRML optimizar el renderizado. La información de este nodo es editable mediante el *Edit Shape Hints Dialog Box.*

23. Esfera

Nodo que representa una esfera. Por defecto, la esfera es centrada en (0, 0, 0) y tiene un radio de 1. La información es ingresada mediante el *Edit Sphere Dialog Box.*

24. Faro de luz

Define una fuente de luz tipo foco. Un faro de luz es ubicado en 3D e ilumina en forma de cono en una dirección en particular. La información es ingresada mediante el *Edit Spot Light Dialog Box.*

25. Interruptor.

Es un nodo de grupo que atraviesa uno, ninguno o todos sus hijos. Este es usado para proporcionar efectos de encendido o apagado entre diferentes propiedades. El campo *whichChild* especifica el índice del hijo que atraviesa, donde el primer hijo tiene índice 0. Un valor de -1 indica que no atraviesa a ninguno. Un valor de -3 atraviesa a todos. La información es ingresada a través del *Edit Switch Dialog Box.*

26. Textura.

Define un mapa de textura y sus parámetros. Indica la secuencia de las formas para una textura. La información para el nodo es editada a través del *Edit Texture Dialog Box.*

27. Transformación de Textura.

Define una transformación geométrica de la textura y consiste de una transformación no uniforme a partir de un punto arbitrario, una rotación y una traslación. La información del nodo es editada mediante el *Edit Texture Transform Dialog Box.*

28. Separación de transformación

Es un nodo de grupo similar al separador de nodo. Almacena el estado de la escena antes de transformarse al atravesar los hijos para luego restaurarlos. La información es ingresada mediante el *Edit Transform Separator Dialog Box.*

29. Traslación.

Define la traslación mediante un vector 3D.La información del nodo de traslación es editada en el *Edit Translation Dialog Box.*

4. CONSTRUYENDO UN EJEMPLO EN V-REALM

Se presenta un ejemplo simple para crear una esfera. Haremos cambios a la escena gráfica para aumentar el tamaño de la esfera (transformación). Sin embargo, la esfera podría ser realzada también mediante algunas traslaciones, cambios de apariencia o manipulación de luces.

```
#VRML V1.0 ascii
Separator {
        Sphere {
                radius  1
        }
    }
```

Un ejemplo de mejoramiento mediante la suma de textura al objeto es el siguiente.

```
#VRML V1.0 ascii
Separator {
        Texture2 {
                filename

"C:\PROGRA~1\VRBUILD\PROGRAM\Texture\wood.gif"
                image   0 0 0
                wrapS   REPEAT
                wrapT   REPEAT
        }
        Sphere {
                radius  1
        }
    }
```

Un ejemplo para agregar una fuente de luz a la escena, sería:

```
# VRML V1.0 ascii
Separator {
    PointLight {
        on TRUE
        intensity    1
        color    1 0 0
        location      0 0 1
    }
    Texture2 {
        filename
"C:\PROGRA~1\VRBUILD\PROGRAM\Texture\wood.gif"
        image   0 0 0
         wrapS REPEAT
        wrapT  REPEAT
    }
    Sphere {
            radius  1
    }
}
```

Agregando una transformación, el listado sería:

```
# VRML V1.0 ascii
Separator {
    PointLight {
            on  TRUE
            intensity    1
            color    1 0 0
            location      0 0 1
    }
    Texture2 {
            filename
"C:\PROGRA~1\VRBUILD\PROGRAM\Texture\wood.gif"
            image   0 0 0
            wrapS  REPEAT
            wrapT  REPEAT
    }
    Transform {
            translation 1 1 0
            rotation      0 0 1  0
            scaleFactor .5 1.5 .5
    scaleOrientation    0 0 1  3.459
            center   1 1 0
    }
    Sphere {
            radius  1
    }
}
```

La suma progresiva de nuevos nodos de propiedad a la escena gráfica puede ser de los siguientes tipos: luces, cámaras, transformaciones, apariencia, grupo, línea WWW, etc. Los nodos de apariencia requieren de nodos objetos para ser válidos. También, el nodo "ancla" requiere de un nodo objeto (ej. Esfera) para poder funcionar. El usuario debe estar seguro de usar los nodos correctos y en el orden correcto. Por ejemplo, un nodo objeto debe ser declarado antes que uno de textura.

5.3. 3D-WEBMASTER

1. DESCRIPCIÓN DE 3D-WEBMASTER

Este producto de Superscape se compone de los siguientes editores:

a) Editor de formas (*Shape Editor*):
b) Editor de mundos (*World Editor*):
c) Editor de imagen (*Image Editor*):
d) Editor de sonido (*Sound Editor*):
e) Visualizador (Visualizer / Viscape): son dos ambientes que permiten desplegar e interactuar con el mundo virtual que ha sido creado con los otros editores.

Existen otros editores que componen el producto original y que permiten agregar propiedades adicionales. También es importante definir los siguientes conceptos que logran un óptimo entendimiento de este producto. Ellos son:

- Punto de vista: 3D-WebMaster permite ver los mundos construidos desde diferentes posiciones (o cámaras) fijas en proyección polar (perspectiva) o paralela (elevaciones).
- Atributos: corresponden a propiedades que se asocian a un objeto o grupo y que les permiten ciertas alternativas de visualización (ejemplo: ser visible o invisible, rotación, movimiento, etc.).
- Cubo delimitador: cada forma en un mundo virtual es creada dentro de un cubo delimitador, invisible, que permite a VRT ordenar los objetos correctamente en el ambiente diseñado. Este cubo debe ajustarse lo más posible a la forma diseñada, de tal manera que represente su frontera física.
- Movimiento: a los objetos se les puede dar una amplia gama de rasgos dinámicos, semejante al comportamiento de los objetos del mundo real. Estos incluyen movimiento lineal y angular, propiedades de fricción y gravedad, y las interacciones de colisión entre otras más.
- Rotaciones: un objeto puede rotar en cualquier dirección, asignarle velocidad angular y combinación de movimientos lineales y curvos.

2. INTRODUCCIÓN AL SCL

Para crear una aplicación con comportamiento es necesario introducir al lenguaje propio de esta herramienta de desarrollo en RV llamada SCL (*Superscape Control Language*), que es un lenguaje de programación derivado del famoso C++. SCL controla y determina la ejecución de comportamientos de un objeto bajo el concepto de evento-efecto y es ejecutado en tiempo real. Para ello el producto 3D-WebMaster realiza una ejecución permanente del ambiente virtual, esto quiere decir que el software analiza y ejecuta todos los programas (de principio a fin), una cantidad determinada de veces por segundo (por *frame*), por lo que chequea todos los eventos que produzcan un efecto en cada *frame*.

Para desarrollar la aplicación en este capítulo de aprendizaje se presentarán algunas de las instrucciones con las cuales podrá desarrollar una aplicación de mediana complejidad y con las cuales podrá experimentar las posibles inquietudes en el tema. Considerando las siguientes convenciones:

<*ítem*> define un argumento o valor según sea el tipo de su definición
Ejemplo: < *Constante* > : significa que este debe ser reemplazado por un valor numérico constante.

[...] define que su contenido es opcional

Todas las instrucciones deben escribirse en minúsculas.

Algunas Instrucciones:

1. CASE: Permite tener varias alternativas para ejecutar a partir del cumplimiento de una de ellas, y de no cumplirse se realiza el caso indicado bajo "else".

Forma:
switch (<valor numérico>);
case <constante>: <instrucción>
[else
< instrucción>]

2. CHAR: Permite declarar una variable y aceptar como valor un carácter.

3. IF: Permite ejecutar instrucciones si se cumple la condición, de no cumplirse se ejecuta lo indicado bajo "else" si es que está presente.

Forma:
if (<condición>)
<instrucción>
[else
<instrucción>]

4. REPEAT: repite un conjunto de instrucciones mientras se cumpla el número (corresponde a una cantidad de frames).

Forma:
repeat (<número>)
<instrucción>

5. { ... } tiene por función el agrupar un conjunto de instrucciones con la finalidad de asociarlas a una mayor.

Forma:
{ <instrucción 1>;
:
:
<instrucción N>; }

6. DO: repite un conjunto de instrucciones a lo menos una vez hasta que se cumpla la condición.

Forma:
do
<instrucción>
until (<condición>);

7. BREAK: permite salir de un ciclo do o repeat.

Forma: *break;*

8. ACTIVATE: permite que si se "pincha" el objeto que tenga esta instrucción (con el botón izquierdo del mouse), ejecuta un conjunto de instrucciones.

Forma:
activate (me, 0) ;

el valor 0, al reemplazarlo por 13, representa el botón derecho.

Ejemplo:
if (activate (me, 0))
<instrucción>;

9. SIMBOLOS: se utilizan para realizar comparaciones dentro de una condición.

a) > mayor que
b) < menor que
c) <= menor o igual que
d) >= mayor o igual que
e) = = igual que
f) ≠ distinto a
g) && y
h) | | o

Ejemplo:
```
if ( x >= 2 && x<5)
                <instrucción>;
```

10. Operaciones aritméticas básicas:

a) + suma; ejemplo: <valor1> + <valor2>;
b) - resta
c) / división
d) * multiplicación

11. = asignación de un valor a una variable

Forma:
```
<variable> = <expresión>;
```

Ejemplo:
```
X = X + 1;
```

12. ; separador de instrucciones.

Ejemplo:
```
if ( x > 5 )
                {
                        x = x + 1;
                        y = y + 2; }
```

13. INT: declara una variable numérica para valores entre -32767 a +32767.

Forma:
```
int <variables>;
```

Ejemplo:
```
int x, y, z;
```

14. FLOAT: declara una variable numérica para valores entre -10e38 a 10e+38.

Ejemplo:
```
float x;
```

15. XPOS, YPOS, ZPOS permiten mover un objeto en los ejes x, y, z si tiene las propiedades de posición inicial y dinámica.

Forma:
```
xpos ( <objeto>);
```

Ejemplo:
```
if (x > 20)
                ypos('mueble') += 100;
```

16. X + = 10; Esto es equivalente a x = x + 10;

17. VIS: permite a un objeto invisible ser visible.

Forma:
```
vis (<objeto>);
```

Ejemplo:
```
vis (me); vis ('puerta');
```

18.- INVIS: opuesto a VIS.

19. DIST2: contiene la distancia entre dos objetos.

Forma:
```
x = dist2( <objeto1>, <objeto2>);
```

20. XANGV, YANGV, ZANGV: atribuye velocidad angular a un objeto.

Forma:
```
xangv ( <objeto>)
```

Ejemplo:
```
xangv (me) += 100;
```

3. EDITORES DE 3D-WEBMASTER.

Describiremos en mayor detalle los distintos editores del programa, los cuales permiten el diseño de ambientes virtuales. Debemos resaltar que los editores de mayor importancia son el Editor de Formas (SHAPE EDITOR) y el Editor de Mundos (WORLD EDITOR), con los cuales se puede crear una aplicación de suficiente calidad. Sus otros editores permiten incorporarles carácterísticas que mejoran la aplicación.

a) Editor de Mundos: este editor incorpora funciones de acceso inmediato a las que mediante iconos y menús permiten la rápida construcción de mundos virtuales interactivos. Estas funciones incluyen posicionamiento de objetos, color, iluminación, duplicación de objetos, dinámica, rotación, texturizado, escalamiento de objetos, definición de puntos de vista, animación, entre otros. La edición se realiza de manera interactiva en tiempo real de forma que las modificaciones pueden evaluarse inmediatamente. Además, en este editor se pueden definir comportamientos mediante el lenguaje SCL.

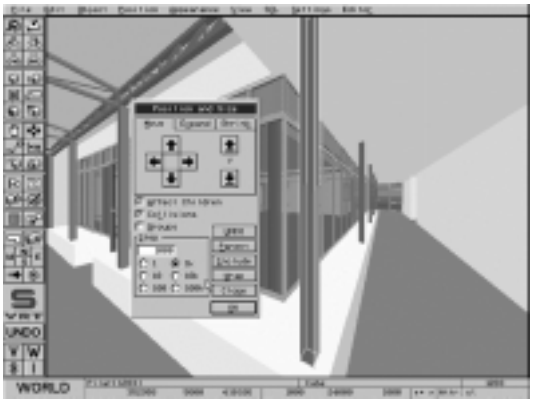


Fig. 5.11

b) Editor de formas: 3D-WebMaster incorpora un modelador completo. Además de permitir la importación de archivos DXF, este editor permite construir los objetos que configuran el mundo virtual. El diseño y la edición de objetos se realiza de forma interactiva en 3D y en tiempo real de manera que las modificaciones se pueden evaluar de forma instantánea. El editor posee funciones tales como transformaciones, giro y extrusiones, malla de polígonos, iluminación, animación de formas, etc.

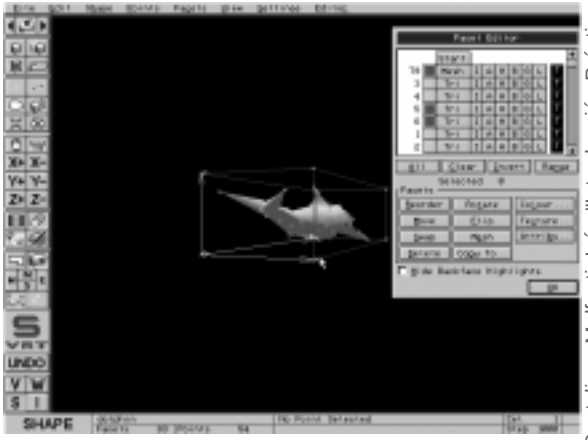


Fig. 5.12

c) Editor de imágenes: Es un editor completo de retoque fotográfico para modificar y editar texturas e incorporarlas en un mundo virtual. Soporta todos los formatos gráficos tales como PCX, JPG, GIF, TIF, BMP, etc. Posee todas las herramientas necesarias para la edición de texturas como pinceles, spray, cortar, pegar, transparencia, relleno, etc. Al estar integrado en el propio Software, el editor supone el ahorro de tiempo en este proceso de alteración de imágenes.

Fig. 5.13

d) Editor de sonidos: El sonido añade una dimensión adicional al mundo virtual. El editor de sonidos permite aplicar sonido a los objetos, incluyendo voz, mono y estéreo, de 16 bits, con ajuste de volumen automático en función de la distancia entre el usuario y la fuente de sonido (objeto). Este editor permite acciones tales como: filtrar, re-escalar, componer y variar el pitch de cualquier sonido. Soporta grabaciones directas de archivos WAV e incluye librerías de éstos. Estos sonidos pueden ser activados usando SCL desde dentro del mundo virtual.

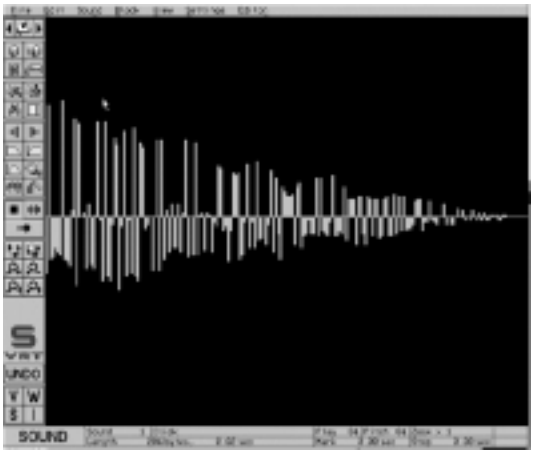


Fig. 5.14

4. PROGRAMACION

La programación de comportamiento de un objeto dentro de una determinada aplicación permite que dicho mundo se aproxime a la realidad. Para lo anterior se debe indicar que un objeto es "programado" dentro del Editor de Mundos y dicho objeto debe poseer los atributos que le permitan la dinámica de movimiento que se desea. Con el objeto de comprender este proceso se presentan algunos ejemplos simples que determinan un comportamiento básico de los objetos. Los atributos de un objeto deben ser asignados para que esté habilitado y poder efectuar el comportamiento solicitado por el programa de comportamiento. Ejemplo, para que un objeto pueda rotar debe poseer el atributo "r".

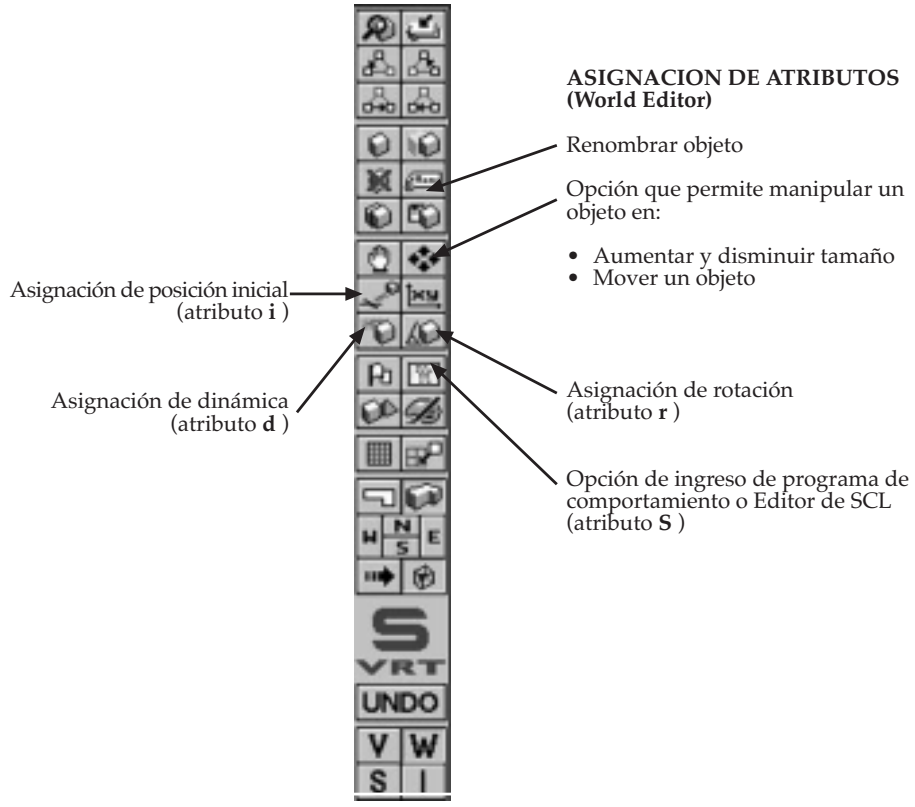


Fig. 5.15

Al seleccionar el Editor de SCL, se podrá incorporar del código de programación de comportamiento de un objeto. El proceso es el siguiente:

- Seleccionar el objeto para poseer el comportamiento.
- Asignarle los atributos necesarios.
- Incorporar código de comportamiento.
- Ir a Viscape para ejecución.

Fig. 5.16 - Editor SCL

Para comprender un programa de comportamiento se dan los siguientes ejemplos.

Ejemplo 1:

1. *Crear un Objeto nuevo (copia del objeto base)*
2. *Incorpore propiedades al nuevo objeto:*
   r - i
3. *Ingrese el siguiente programa sobre obj. nuevo:*
   if (activate(me, 0))
   yrot(me)+=5;
4. *en Viscape presione botón izq. del Mouse sobre objeto nuevo.*

- Este ejemplo permite que un objeto rote al ser tocado por el mouse y a su vez se presione el botón izquierdo haciendo click sobre dicho objeto.

Los atributos necesarios son:

r : rotación.

i : posición inicial.

Ejemplo 2

1. *Renombre a objeto nuevo como "otro"*
2. *Agregue propiedad a "otro":*
   i - d (quite gravedad)
3. *Ingrese el siguiente programa al objeto base:*
   if (activate(me, 0))
   ypos('otro')+=100;
4. *en Viscape presione botón izq. del Mouse sobre objeto base.*

- Este ejemplo permite a un objeto moverse en forma horizontal al ser tocado, el objeto base, por el mouse, y a su vez se presione el botón izquierdo haciendo click sobre dicho objeto.

  Los atributos necesarios son:

  r : posición inicial.

  d : dinámica

  Para quitar gravedad debe colocar en 0 dicha propiedad en dinámica (el incorporar atributo d).

Un atributo muy utilizado es "a" de velocidad angular, es decir, un objeto debe poseer este atributo para permanecer en movimiento giratorio indefinidamente. Este es asignado seleccionando el objeto que girará o luego desde el menú (barra superior de WebMaster) se selecciona:

Position -> Angular Velocity

Ejemplo 3

Este ejemplo se basa en el prototipo realizado para simular una máquina puesto que a través de ella se puede validar con mayor exactitud la interacción de sus diferentes componentes y su trabajo como objetos independientes y su interacción en el tema de entrenamiento.

Fig. 5.17

Dicha máquina se encuentra ubicada físicamente en la Universidad del Bío-Bío (Depto. de Maderas). Está constituida por piezas que poseen movimiento, pero que no se detallarán por no ser de incidencia en la máquina.

La construcción comienza con el diseño de la base de la máquinas la cual se estructura a partir de la *Shape* (forma) elemental del editor de formas (un cubo). Este último se duplica cuantas veces se requiera y se modifican sus tamaños para lograr las diferentes partes de dicha base. Luego se procedió a ordenar las superficies de los objetos, de forma de evitar superposiciones de las mismas. Las figuras circulares se diseñan bajo el concepto de sólido de revolución, es decir, se genera una línea en el plano y luego se hace rotar sobre uno de los ejes.

A medida que se van generando las formas, se van incorporando al mundo virtual (máquina) a través del Editor de Mundos. En este último se procede a darle el tamaño adecuado a cada pieza y el colorido correspondiente. En este mismo editor se le crea el programa de comportamiento asociado al objeto que lo requiera. Ejemplo, giro de una rueda al presionar un botón de arranque.

Se debe destacar que el programa debe ir asociado, por recomendación, en el objeto que activa a otro, por ejemplo el botón de arranque.

Una demostración de programa (para el caso del botón) sería:

```
Resume (0,2);
if (activate(me,0))
{
        do
                { zrot('rueda') += 5;
                waitf;
                }
        until (activate(me,0))
}
```

Este programa se graba (asocia) al objeto 'BOTON' y al ser activado hará que el objeto 'RUEDA' rote 5º en el eje Z.

Para entenderlo en todo su contexto, es necesario explicar que un mundo virtual se presenta a la vista como una cinta de película; es decir todo movimiento que ocurra se realiza en un cuadro o foto (*frame*) de la cinta , esto último se realiza en tiempo real, por lo tanto no sería perceptible a la vista (entendamos que la película corre aproximadamente a 24 cuadros por segundo). Además todos los programas asociados a los objetos son ejecutados a la vez (en un cuadro). En el caso de un cuadro (*frame*), de este software, el tiempo es de 1/60 seg. cada cuadro, es decir casi 4 veces más rápido que una película.

Teniendo en consideración lo anterior, el programa ejemplo funcionaría de la siguiente forma:

RESUME(0,2), permite reconocer la ubicación del WAITF. Para el primer dígito, 0 implica WAITF y 1 implica WAITFS . El segundo dígito (valor 2) indica que el WAITF (o WAITFS, según sea el caso) se encuentra dentro del segundo {, anidado.

WAITF permite que las acciones antes de él se ejecuten en el primer frame y las restantes instrucciones en el segundo. WAITFS(5) indica una espera de 5 frames.

ACTIVATE(ME,0) es verdadero si se activa el objeto al cual pertenece el programa (ME), con el mouse. El 0 indica que la activación debe realizarse con el botón izquierdo del mouse, 13 corresponde al botón derecho.

La instrucción DO se ejecuta hasta que se cumpla el UNTIL.

ZROT('RUEDA') += 5, permite rotar al objeto 'RUEDA' en 5 grados en el eje Z. Es así que el giro será de 5º por frame y se hará al activarse el botón de arranque hasta que se vuelva a activar (todo esto a través del mouse).

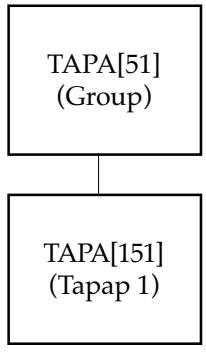
Se pueden realizar programas más complejos que aumenten el realismo de los objetos.

La descripción de un objeto es mediante:

Nombre de objeto corresponde al nombre con el cual es reconocido en el *Word Editor*. Tipo corresponde al nombre de la FORMA que le corresponde en el *Shape Editor*. El objeto lleva "*" en la parte superior si es que posee programa de comportamiento.

Dos ejemplos de esto es:

Objeto TAPA[51]



Objeto TAPA[51]
Tipo: Grupo
Hijos: TAPA[151]

Shape Tapap 1

Fig. 5.18 - Máquina (grupo).

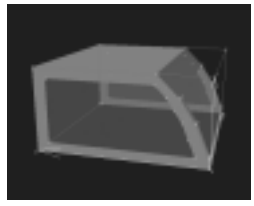
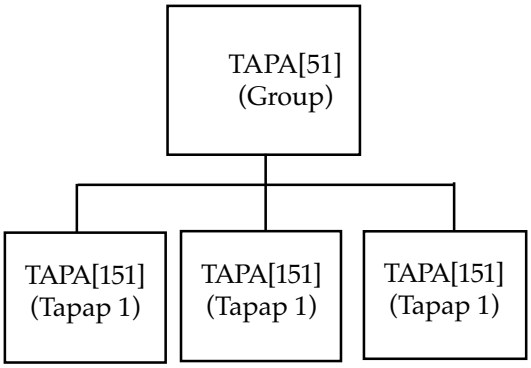


Fig. 5.19 - Cubierta (hijo).

Objeto Gpe1[226]



Objeto Gpe1[226]
Tipo: Grupo
Hijos: Sal1[227]
Gir[228]

Fig. 5.20

Shape cube[9]

Fig. 5.21

Shape Cilindro

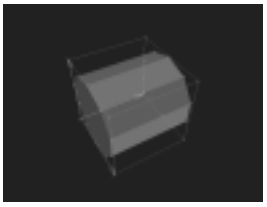
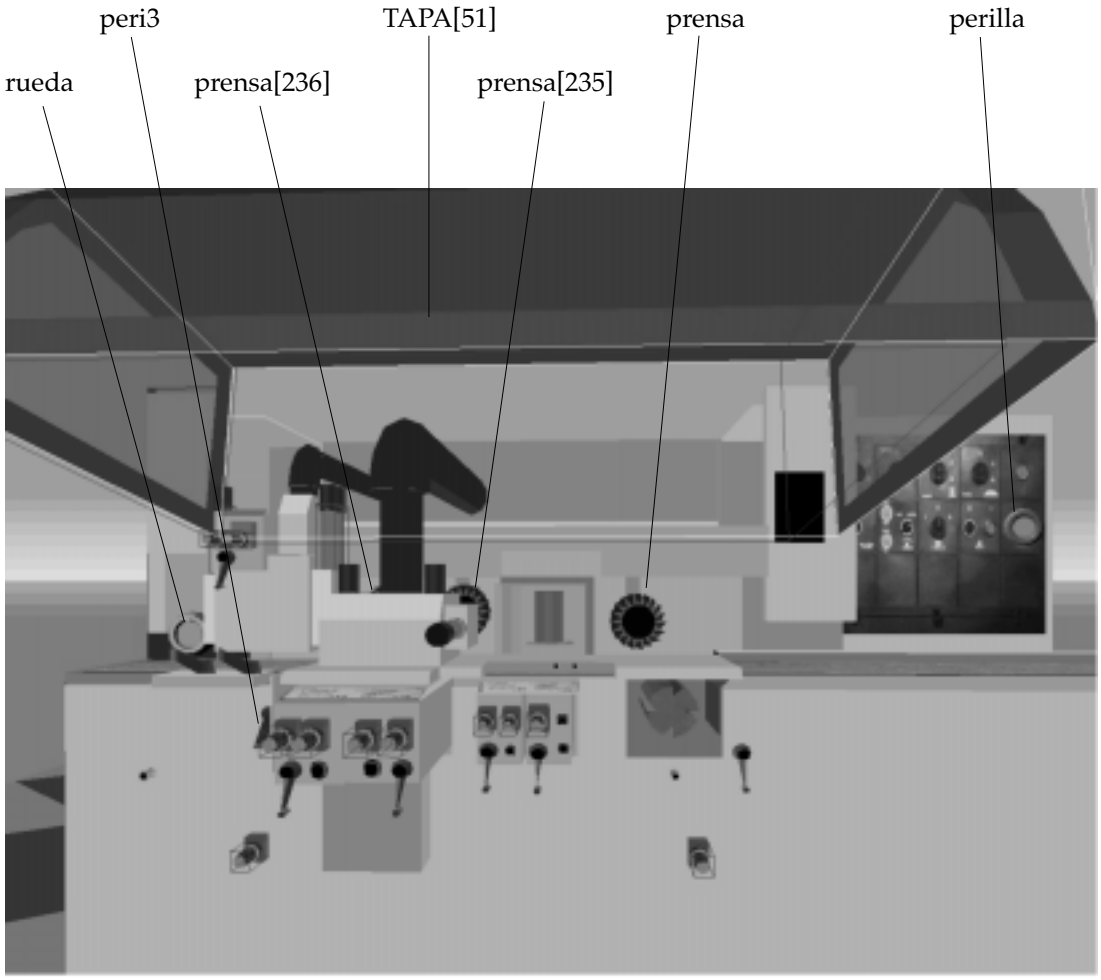


Fig. 5.22

Aplicando este procedimiento en todos los objetos para constituir el ambiente virtual de esta máquina se obtiene el diseño virtual siguiente, en el cual detallamos algunos programas que permiten comprender su actuación.



5.23 - Distribución de componentes.

Los programas y descripciones siguientes hacen referencia a los objetos identificados en esta máquina. Se debe señalar que muchos de los objetos presentes en dicha máquina poseen el mismo comportamiento por lo tanto el mismo programa. Generalmente, se programa un objeto y se realiza una copia del código en los otros objetos que tienen la misma acción por realizar. Asimismo, tenemos nombres tales como TAPA[51], esto se debe a que generalmente se diseña un objeto (ejemplo, TAPA) y a éste se le realizan tantas copias como se desee, las cuales se van denominando TAPA[1], TAPA[2], etc.

Programa 1

```
Resume (0, 1);
if (activate(me,0))
{
zangv('prensa')=9;
zangv('prensa[235]')=9;
zangv('rueda')=9;
zangv('prensa[236]')=9;
}
if (activate(me,13))
{
zangv('prensa')=0;
zangv('prensa[235]')=0;
zangv('rueda')=0;
zangv('prensa[236]')=0;
}
```

Objeto que posee el programa:

Perilla

Descripción:

Este programa, al hacer click con el botón izquierdo del mouse (valor "0" junto a "me") sobre "Perilla" (denominado "me") hace girar indefinidamente los objetos: "prensa", "prensa[235]", "rueda" y "prensa[236]" en 9 grados por frame. Al realizar la acción anterior con botón derecho (valor 13) todos los objetos afectados se detienen (valor 0 para "zangv").

Atributos de los objetos que se mueven:

"a", "r"

Programa 2

```
Resume (0, 1);
if (activate(me,0) && xrot(me)<80)
xrot(me)+=10;
if (activate(me,13) &&
xrot(me)> - 80)
xrot(me)-=10;
```

Objeto que posee el programa:

peri3

Descripción:

Este programa, al hacer click con el botón izquierdo del mouse sobre "peri3" y mientras el ángulo de giro (del mismo objeto) es menor a 80°, lo hace girar inmediatamente 10° hacia la derecha (función "xrot"). Al realizar la acción anterior con botón derecho (valor 13) gira a la izquierda.

Atributos del objeto que se mueven:

"r"

Programa 3

```
Resume (0, 1);
if (activate(child(me),0) &&
xrot('TAPA[51]')> -100)
xrot('TAPA[51]')-=10;
if (activate(child(me),13) &&
xrot('TAPA[51]')<0)
xrot(me)+=10;
```

Objeto que posee el programa:

TAPA[51]

Descripción:

Como TAPA[51] es un Grupo, mediante "child" se está haciendo referencia a los objetos agrupados por TAPA. Este programa, al hacer click con el botón izquierdo del mouse y si el ángulo no ha excedido -100°, sobre cualquier objeto incluido por TAPA[51], hace girar inmediatamente al grupo TAPA[51] en 10° hacia arriba (sube la cubierta verde de la máquina). Al realizar la acción anterior con botón derecho y el ángulo no es inferior a 0 (línea horizontal), el grupo TAPA[51] gira en el sentido contrario al anterior (acción de bajada de la cubierta).

Atributos de los objetos que se mueven:

"r"

Nota: Como el objeto que se mueve es el mismo que posee el programa, "me" es equivalente a "TAPA[51]".