Programación de gráficos 3D con Mathematica, DrawGraphics, CurvesGraphics, LiveGraphics3D y JavaView

Figueroa, Geovanni. Mora, Walter
Escuela Matemática
Instituto Tecnológico de Costa Rica

Resumen:

Se muestra como integrar las herramientas: *Mathematica* (y los paquetes DrawGraphics y CurvesGraphics), LiveGraphics3D, JavaView y html, para crear algunas figuras 3D las cuales se pueden incrustar en páginas Web independientes y con posibilidad de interacción.

Palabras claves: curvas, superficies, parametrización, *Mathematica*, liveGraphics3D, JavaView, visualización 3D, DrawGraphics, CurvesGraphics.

Introducción

Las representaciones visuales siempre han sido fundamentales en la exploración, comprensión y transmisión de conceptos de la matemática. La visualización 3D, en cálculo en varias variables o geometría del espacio, han estado restringida –en Internet- en su mayoría a figuras muy sofisticadas pero estáticas, o bien, a figuras con cierto grado de interacción, que usualmente sólo incluye rotación. Como es usual, la creación de situaciones de aprendizaje con cierto grado de interactividad requiere la integración de varias herramientas.

El objetivo de esta comunicación es mostrar como integrar *Mathematica*, LiveGraphics3D, JavaView y html, para crear algunas figuras 3D incrustadas en páginas Web independientes con posibilidad de interacción, en este caso rotación.

En particular vamos a:

- mostrar las acciones básicas que se deben seguir con el objetivo de instalar LiveGraphics3D y JavaView,
- construir páginas Web interactivas con gráficos desarrollados en *Mathematica* ,
- construir superficies, sólidos y curvas con las herramientas de DrawGraphics y CurvesGraphics
- analizar algunas técnicas para la construción de superficies que requieren el uso de las herramientas de programación de *Mathematica*, con métodos generales.

Dejamos para una futura comunicación, la manera de interactuar con los objetos geométricos 3D a través de la definición de variables independientes y dependientes (gráficos con parámetros), las cuales se pueden hacer variar arrastrando puntos, como se haría en Cabri o en Sketchpad.

No es necesario tener una gran experiencia en programación para comprender la implementación de los ejemplos que se muestran. Todos los ejemplos fueron desarrollados con *Mathematica* 5.0 sobre Windows XP. Las imágnes fueron generadas con *Mathematica* 5.0, en formato EPS, y editadas con Adobe Illustrator y Fireworks.

Primitivas, Mathematica, LiveGraphics 3D y Jview

Los gráficos 3D se construyen como un conjunto de primitivas: puntos, líneas y polígonos. Además, están presentes propiedades de las primitivas como: color, grosor, etc. Por ejemplo, la figura 1. muestra un dodecaedro.

Figura 1. Dodecaedro, se agregó transparencia con Adobe Illustrator

Parte del código interno generado por *Mathematica* para construir el objeto 3D es:

```
...
AbsoluteThickness[1],
RGBColor[0.670, 0.732, 0.896],
Polygon[{
         {141.875, 93.9375},
         {174.438, 129.688},
         {234.938, 122.688},
         {141.875, 93.9375}
       }],
GrayLevel[0.000],
AbsoluteThickness[0.5],
Line[{
      {141.875, 93.9375},
      {174.438, 129.688},
      {234.938, 122.688}
    }],
RGBColor[0.889, 0.750, 0.702],
Polygon[{
         {174.438, 129.688},
         {141.875, 93.9375},
         {116.438, 196.625},
         {174.438, 129.688}
       }],
GrayLevel[0.000],
Line[{
      {174.438, 129.688},
      {141.875, 93.9375}
    }]...}
```

En este código observamos la presencia de:

- primitivas como: `Polygon[] y Line[]`.
- propiedades como: `RGBColor[0.889, 0.750, 0.702]`, `AbsoluteThickness[0.5] .`

La idea básica que esta detrás de la creación de figuras 3D interactivas, es la de generar el código de un objeto geométrico con *Mathematica* y luego prepararlo para que sea usado por Livegraphics3D y/o JavaView.

Livegraphics3D y JavaView

Livegraphics3D (LG3D) es un interpretador de gráficos gratuito, implementado en Java, que permite desplegar y rotar gráficos 3D generados en *Mathematica* . Las clases Java que lo componen (live.jar, 93 KB) reciben el código de los gráficos, de una manera adecuada, y lo pueden presentar en una página Web si el navegador tiene la máquina virtual de Java habilitada.

Livegraphics3D además de ofrecer la opción de rotación, zoom, etc., también permite gráficos con parámetros, es decir, ofrece la posibilidad de arrastrar puntos con el mouse para lograr interacción con los componentes del gráfico, como se haría en Sketchpad o Cabri. LG3D es un applet *liviano*, con una pequeña desventaja: el algoritmo que decide que primitivas van adelante y cuales primitivas van detrás en una vista, es sencillo y rápido, pero no maneja muy bien esta faceta.

Jview un interpretador de gráficos gratuito, implementado en Java, que permite desplegar y rotar gráficos 3D generados en *Mathematica* o Maple. Las clases Java que lo componen (javaview.jar, 696 KB; jvLite.jar, 188 KB; jvx.jar, 622 KB; vgpapp.jar, 425 KB) reciben el código de los gráficos, de una manera adecuada, y lo pueden presentar en una página Web siempre y cuando que el navegador tenga habilitada la mquina virtual de Java.

JavaView tiene gran cantidad de facetas para interactuar con los gráficos. Se puede utilizar las clases javaview.jar (696 KB) o las clases jvLite.jar (188 KB) para levantar los gráficos en una página Web. La diferencia entre estos dos paquetes de clases es, además del tamaño de las clases, el acceso a los paneles de control (aunque se conservan algunas propiedades que pueden ser modificas con teclas). La calidad de los gráficos con JavaView es excelente, de hecho podemos incluir transparencia y textura. JavaView no tiene una manera fácil y visible de trabajar con gráficos con parámetros como LG3D.

LG3D y JavaView entienden la mayoría de las funciones de *Mathematica* y se pueden usar de manera integrada con este software. Ambos ofrecen clases públicas para trabajar directamente en Java. Aquí vamos a trabajar con ambas opciones.

Antes de continuar, debemos instalar las herramientas necesarias para implementar los gráficos

Instalación

Para construir nuestros gráfcos 3D vamos a necesitar

- Java: para integrar applets de Java con *Mathematica* (J/Link). Si no tiene Java, descargue e instale el JSDK desde el sitio de Sun Microsystems ([10]). No olvide habilitar java en su navegador.
- *Mathematica*: para generar el código de los gráficos ([7]).
- LiveGraphics 3D (LG3D): para interpretar el código y levantar el gráfico. Puede ser bajado del sitio de LiveGraphics3D ([8])
- JavaView: para interpretar el código y levantar el gráfico. Puede ser bajado del sitio de JavaView ([9])
- Un editor de texto: para editar archivo html, podría ser el block de notas.

Para poder usar LG3D y JavaView, más que una instalación, debemos copiar algunos archivos en el lugar apropiado.

Asuminos la instalación para el caso de `Windows XP` y *Mathematica* 5.0. La instalación en otro sistema operativo y otra versión de *Mathematica*, debería ser análogo. Suponemos que *Mathematica* fue instalado en el directorio por defecto, es decir,

`C:/Archivos de programa/Wolfram Research/Mathematica/5.0`,

Bajo estas circunstancias la instalación sería así:

1. LiveGraphics 3D
    1. Descargar `LiveGraphics3D.m` y `live.jar` desde sitio Web de LG3D ([8]).
    2. Copiar el archivo `LiveGraphics3D.m` en el directorio

        `C:/.../Wolfram Research/Mathematica/5.0/AddOns/StandardPackages/Graphics/`

    3. `live.jar` lo podemos copiar, por ahora, en la misma carpeta en la que estará la página web que contedrá los gráficos.

        

        Figura 2.

    4. Sería conveniente copiar la carpeta `Graphics` en

        `C:/Archivos de programa/Wolfram Research/Mathematica/5.0/AddOns/AutoLoad`
        si queremos que el paquete se cargue automáticamente cada vez que abrimos *Mathematica* . En todo caso, no vamos a asumir que la carpeta `Graphics` está en `AutoLoad`

2. JavaView
    1. Descargar `jv_math.zip` (para integrar *Mathematica* y JavaView) desde el sitio Web de JavaView ([9]).
    2. Descomprimir el archivo `jv_math.zip` en el directorio

        `C:/.../Wolfram Research/Mathematica/5.0/AddOns/Applications`

        de tal manera que quede
        `C:/.../Wolfram Research/Mathematica/5.0/AddOns/Applications/JavaView`

    3. Tal y como se indica en la página de descarga, el usuario debe registrarse para que le envién un archivo registro llamado `jv-lic.lic` el cual debe copiarse en la carpeta `JavaView/rsrc`. Esto evitará ciertos mensajes no deseados en nuestros gráficos.
    4. De la carpeta `JavaView` debe copiar la subcarpeta `Java`. Esta carpeta, renombrada como `jars`, la pegamos en las carpetas en donde vamos a poner las páginas Web que contiene nuestros gráficos.

        

        Figura 3.

        La subcarpeta `Java` se renombra `jars` solo por comodidad, dado que las páginas web generadas por JavaView llaman a los programas JavaView desde una carpeta de nombre `jars`.

LG3D, Jview y *Mathematica*

Algunos objetos geométricos que vamos a usar, se encuentran implementados en *Mathematica*. Por ejemplo, el dodecaedro de la figura 1, se construye así

`In[1]:= « Graphics`Graphics`` incluir 'Graphics' (si no esta en AutoLoad)

```
In[2]:= g =
Show[Polyhedron[Dodecahedron]
,Boxed ->False]
```

Muchos objetos geométricos los tendremos que implementar usando las herramientas de programación de *Mathematica*.

LG3D y *Mathematica*

Una vez que tenemos el objeto geométrico `'g'` implementado en *Mathematica*, usamos la función especial `WriteLiveForm[]`, del paquete `LiveGraphics3D.m`, para convertir el código del gráfico en un código apropiado para que el applet `LiveGraphics3D` lo pueda interpretar y levantar en una página Web. A este archivo le damos un nombre, digamos `dode.m'`

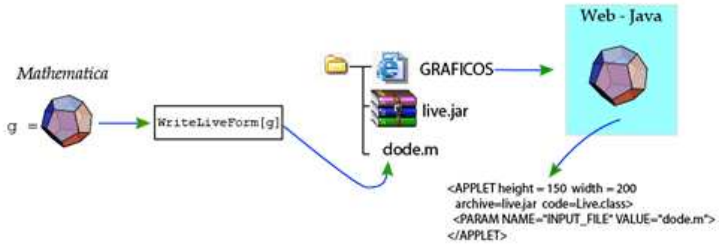

Figura 4.

Para poner el dodecaedro en una página Web, que estará en la carpeta existente

```
E:/MisLG3D
```

ejecutamos el siguiente código en *Mathematica*

```
<< Graphics`LiveGraphics3D`;                          (*paquete
                                                      LiveGraphics*)
g = Show[Polyhedron[Dodecahedron],
         Boxed->False];
WriteLiveForm["dode.m", g, Dir ->"E:\\MisLG3D\\"];
```

Esto escribe el archivo `dode.m` en la carpeta `E:/MisLG3D`. Suponiendo que en esta carpeta ya está presente el archivo `live.jar`, podemos crear una página Web para hospedar el gráfico con el siguiente código:

```
<HTML><HEAD></HEAD> <BODY>
<APPLET height=150  width=200
archive="live.jar" code="Live.class">
<PARAMNAME="INPUT_FILE" VALUE="dode.m">
</APPLET>
</BODY></HTML>
```

[Ver página]

Nota 1: si usa el bloc de notas para crear esta página Web, no debe guardar el archivo como un archivo ".txt", sino como archivo general con extensión ".html"

Nota 2: Puede crear la página web directamente en `E:\\MisLG3D\\`. Podemos crear una página web llamada, digamos `'grafico1.html'`, con el siguiente código

```
WriteLiveForm["dode.m", g, Dir ->"E:\\MisLG3D\\"];

SetDirectory["E:\\MisLG3D\\"];
strm = OpenWrite["grafico1.html"]; (*abre un canal de escritura*)

                                   (*código de la página Web*)
pagina = "<HTML><HEAD></HEAD> <BODY><APPLET height=150
width=200 archive=live.jar code=Live.class>
<PARAM NAME=INPUT_FILE VALUE=dode.m></APPLET>
</BODY></HTML>";
```

```
WriteString[strm, pagina];          (*escribe en el archivo*)
Close[strm];                        (*cierra el canal*)
```

Una vez que tenemos la página Web, podemos aplicar las siguientes acciones al applet

1. Arrastre el mouse para rotar la figura.
2. Zoom = Shift + arrastre vertical

- Nota: En el sitio Web de LG3D ([8]) se indica como integrar LG3D y *Mathematica* . Aquí solo vamos a ver el procedimiento de integración de JavaView y *Mathematica* , en la siguiente sección.

JavaView y *Mathematica*

Una vez que tenemos el objeto geométrico `'g'` implementado en *Mathematica*, podríamos usar la función especial `WriteHtml[ ]`de `JavaView`, para generar una página Web con el gráfico `'g'`.

También podríamos integrar `JavaView` con *Mathematica* y usar el menú de la ventana `JavaView` para hacer esto mismo, solo que con más opciones.

Veremos ambas opciones en el siguiente ejemplo. Como antes, suponemos que queremos poner la página Web en la carpeta

```
E:/MisJview
```

y también suponemos que en esta carpeta está la subcarpeta `jars`.

Para generar la página Web con el dodecaedro, ejecutamos las siguientes líneas de código (cada línea en una celda aparte)

```
Needs["JLink`"] (*conexión Java - Mathematica *)
InstallJava[]
<<JavaView`JLink`; viewer = InstallJavaView[];
g = Show[Polyhedron[Dodecahedron], Boxed ->False];
```

Aquí tenemos dos opciones:

1. Generar una ventana `JavaView` y crear desde ahí la página Web con el dodecaedro
   a. Para esto, se debe ejecutar el siguiente código

```
JavaView[g];
```

La ventana `JavaView` que se genera, tiene la opción de guardar el gráfico en una página Web

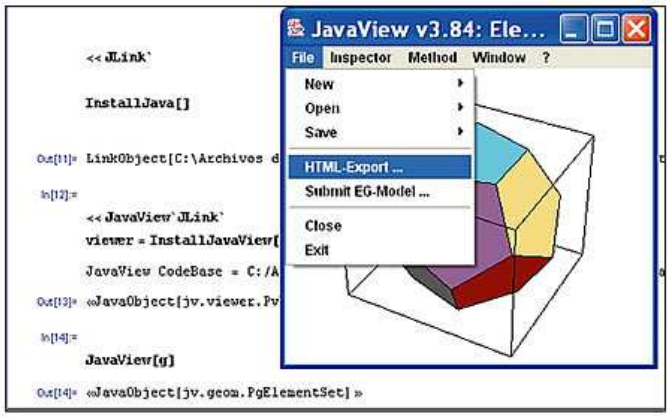

Figura 5.

2. Crear la página con `WriteHtml[]`

   En este caso, ejecutamos las siguientes líneas de código

   1. `In[10]:=SetWorkingPath["E:/MisJview/"];`

2. `In[11]:= WriteHtml[g, "dode.html"]`

- Adicionalmente, se debe abrir la página de `dode.html`, con el bloc de notas u otro editor, y cambiar el código `archive="Java/javaview.jar"` por `archive="jars/javaview.jar"`

jvLite.

`jvLite` es una versión más liviana de JavaView. No incluye los paneles de control pero persisten la calidad de los gráficos y las funcionalidades deseables.

En nuestro ejemplo anterior, para usar `JvLite.jar` en vez de `javaview.jar`, se debe abrir la página `dode.html`, con el bloc de notas u otro editor, y cambiar únicamente el código

```
<APPLET code=javaview.class name="Parameter Demo"
   height=300 width=400
   archive="jars/javaview.jar">
```

por

```
<APPLET code="jvLite.class"  name="Parameter Demo"
   height=300 width=400
   archive="jars/jvLite.jar">
```

Una vez que tenemos la página Web, podemos aplicar las siguientes acciones al applet

1. Arrastre el mouse para rotar la figura.
2. Zoom = Tecla S + movimiento vertical del mouse
3. Una rejilla del PlanoXY se obtiene con Shift + G (de igual manera de deshabilita)
4. Para ver el gráfico con calidad mejorada, usar Shift+S (de igual manera se deshabilita)

Transparencia

Para agregar transparencia a un gráfico 'g', con JavaView, se usa el código

```
geom = JavaView[g];          (* geom es un objeto Jview *)
geom@showTransparency[True];
geom@setTransparency[t];     (* aplicamos transparencia a geom *)
                             (* t se puede ver como porcentaje
                                de transparencia si 0 < t < 1 *)
geom@update[geom]            (* actualiza *)
```

Al crear una página Web con este gráfico, se generan dos archivos adicionales de extensión `.jvd` y `.jvx`. Se puede usar jvLite para levantar este gráfico.

Por ejemplo, considere el gráfico de la figura (se usó 0.8 para la transparencia.)

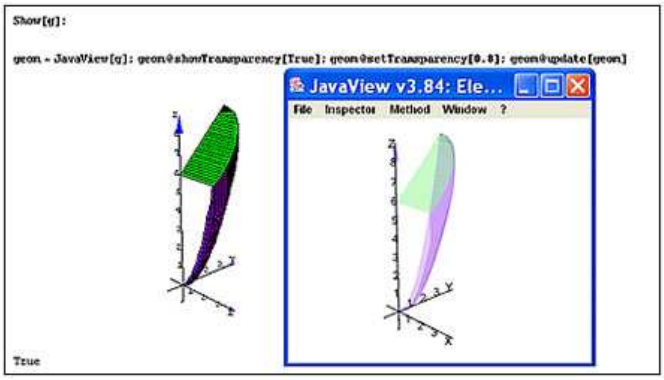

Figura 6.

Implementación de Superficies 3D con Mathematica

En esta sección vamos a implementar gráficos 3D usando el paradigma estándar de *Mathematica*, usando primitivas directamente. Algunas de estas construcciones las vamos a implementar de nuevo dentro del paradigma de DrawGraphics lo que, en algunos casos, resultará en una implementación mucho más sencilla y natural. Sin embargo debemos hacer una

comparación de los gráficos para tener una mejor visión del alcance de las herramientas.

Para poder implementar superficies especiales, usando mayormente primitivas, describimos estas superficies con `Polygon, Line, Point, etc.,` a base de una o varias parametrizaciones. Eventualmente, usaremos `ParametricPlot3D`

Ejes3D y Arrow3D

Para graficar flechas 3D, vamos a usar una pequeña modificación del paquete `Arrow3D` de Harry Calkins

Vamos a crear una sencilla función `Flecha3D[]` la cual pondremos en una *celda* que luego vamos a ejecutar.

Esta función dibuja un cono al final de un segmento que va de `pt1` a `pt2`, el tamaño del cono es un porcentaje de la longitud del segmento. Como el cono se construye con triángulos, se debe indicar el número de triángulos `'nlds'`

El código de la función `Flecha3D[]` es

```
Flecha3D[pt1_, pt2_, nlds_, porc_, color_] := (

      {aa, bb, cc} =pt2-pt1;
      (*construcción de una base ortonormal pata la cabeza de la flecha*)

      nrm1 = {0, 1, 0};
      nrm2 = {1, 0, 0};

      Which[
        aa == 0 && bb == 0, nrm1 = {0, 1, 0}; nrm2 = {1, 0, 0},
        aa == 0 && cc == 0, nrm1 = {1, 0, 0}; nrm2 = {0, 0, 1},
        bb == 0 && cc == 0, nrm1 = {0, 0, 1}; nrm2 = {0, 1, 0},
        aa == 0,
        nrm1 = {1, 0, 0};
        nrm2 = Cross[nrm1, {0, bb, cc}/Norm[{bb, cc}]], bb == 0,
        nrm1 = {0, 1, 0};
        nrm2 = Cross[nrm1, {aa, 0, cc}/Norm[{aa, cc}]], cc == 0,
        nrm1 = {0, 0, 1};
        nrm2 = Cross[nrm1, {aa, bb, 0}/Norm[{aa, bb}]], True,
        nrm1 = {-(((bb + cc)*Abs[aa])/(aa*Sqrt[2*aa^2 + (bb + cc)^2])),
            Abs[aa]/Sqrt[2*aa^2 + (bb + cc)^2],
            Abs[aa]/Sqrt[2*aa^2 + (bb + cc)^2]};
        nrm2 = {((bb - cc)*Abs[aa])/Sqrt[2*aa^4 + (bb + cc)^2*(bb^2 + cc^2) +
                  aa^2*(3*bb^2 + 2*bb*cc + 3*cc^2)], -(((aa^2 + cc*(bb + cc))*
                  Abs[aa])/(aa*Sqrt[2*aa^4 + (bb + cc)^2*(bb^2 + cc^2) +
                  aa^2*(3*bb^2 + 2*bb*cc + 3*cc^2)])), ((aa^2 +
                  bb*(bb + cc))*Abs[aa])/(aa*Sqrt[2*aa^4 + (bb + cc)^2*(bb^2 + cc^2) +
                  aa^2*(3*bb^2 + 2*bb*cc + 3*cc^2)])}
        ];

      (* la cabeza es un porcentaje 'hdsz' del segmento pt1 - pt2 *)

      hdsz  = porc;
      hdrad = Sqrt[((pt2 - (hdsz pt1 + (1 - hdsz)pt2)).(pt2 - (hdsz pt1 +
             (1 -hdsz)pt2)))]/5;

      circ1[ttt_] := hdrad  nrm1 Cos[ttt] +
                     hdrad  nrm2 Sin[ttt] + (hdsz pt1 + (1 - hdsz)pt2);
      circ2[ttt_] := (hdvec = pt2 + hdsz[[1]] (pt1 - pt2)/Norm[(pt1 - pt2)];
      hdrad nrm1 Cos[ttt] + hdrad nrm2 Sin[ttt] + hdvec);

      (* dx es número de lados del cono, Pi/nlds da 2nlds lados*)
      dx = Pi/nlds;

      pts = Partition[N[Table[circ1[t], {t, 0, 2 Pi, dx}]], 2, 1];

      (*Solo fin de Flecha*)
      arrowpolys =Map[{EdgeForm[], SurfaceColor[color],
                       Polygon[Flatten[{#, {pt2}}, 1]]} &, pts];

      Return[arrowpolys];);
```

Una vez que tenemos una función para graficar flechas 3D, podemos crear una función sencilla para los ejes 3D.

```
tira[str_] = StyleForm[str, FontSize -> 14];

Ejes3D[xmin_, xmax_, ymin_, ymax_, zmin_, zmax_] := {

      (*Flechas*)
```

```
 Flecha3D[{xmin, 0, 0}, {xmax, 0, 0}, 3, 0.08, GrayLevel[0]],
 Flecha3D[{0, ymin, 0}, {0, ymax, 0}, 3, 0.09, GrayLevel[0]],
 Flecha3D[{0, 0, zmin}, {0, 0, zmax}, 3, 0.09, GrayLevel[0]],
 (*Ejes*)
 GrayLevel[0],
 Line[{{xmin, 0, 0}, {xmax, 0, 0}, {0, 0, 0}, {0, ymin, 0}, {0, ymax,
       0}, {0, 0, 0}, {0, 0, zmin}, {0, 0, zmax}}],

 Text[tira["X"], {xmax, -0.3, 0}],
 Text[tira["Y"], {-0.3, ymax, 0.1}],
 Text[tira["Z"], {0, -0.2, zmax}],

 (* numeros *)
 Table[Text[tira[i], {i, -0.3, 0.1}], {i, 1, xmax - 1}],
 Table[Text[tira[i], {-0.3, i, 0.1}], {i, 1, ymax - 1}],
 Table[Text[tira[i], {-0.3, 0.1, i}], {i, 1, zmax - 1}],

 (* rayitas *)
 Table[Line[{{-0.1, i, 0}, {0, i, 0}}], {i, ymax - 1}],
  Table[Line[{{i, -0.1, 0}, {i, 0, 0}}], {i, xmax - 1}],
  Table[Line[{{0, -0.1, i}, {0, 0, i}}], {i, zmin, zmax - 1}]
 };
```

EJEMPLO

Como ejemplo, vamos a graficar un sistema de ejes y dos vectores. Usamos la función `ve[pto1, pto2]` para graficar el segmento de `pto1` a `pto2` como un vector.

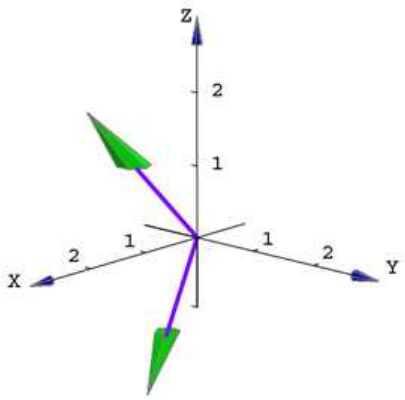

Figura 7.
[Ver en 3D]

El código es

```
color=RGBColor[0, 0.50, 0];
ve[{a1_, a2_, a3_}, {b1_, b2_, b3_}] = {
                    RGBColor[0.50, 0, 1],
                        (*Grosor de la linea*)
                    AbsoluteThickness[2],
                        (*Linea de pto1 a pto2*)
                    Line[{{a1, a2, a3}, {b1, b2, b3}}],
                        (*cabeza de la flecha*)
                    Flecha3D[{a1, a2, a3}, {b1, b2, b3}, 3, 0.4,color]
                                        };

P = {0, 0, 0}; Q = {1, 0, -2}; R = {2, 0, 2};

g = Graphics3D[{
                Ejes3D[-1, 3, -1, 3, -1, 3],
                ve[P, Q],
                ve[P, R]
              }, Boxed -> False,
                 AmbientLight -> RGBColor[1, 1, 0],
                 ViewPoint -> {2.426, 2.190, 0.878}];
Show[g];
```

Nota: Una versión de flecha 3D más sencilla, implementada por Tom Wickham-Jones, se muestra en el código siguiente

---

```
Options[Arrow3D] = {HeadLength -> 0.3, HeadNormal -> {0, 0, 1},HeadWidth -> 0.5};

Arrow3D[a_,b_,opts___]:=
    Module[{abLength=N[Sqrt[(b-a).(b-a)]],abUnit,headPerp,headPerpLength,
             headLength,headNormal, headWidth},

        {headLength,
        headNormal,
        headWidth}={HeadLength,HeadNormal,HeadWidth}/.{opts}/.Options[Arrow3D];


        abUnit=(b-a)/abLength;
        headPerp=Cross[abUnit,N[headNormal]];
        headPerp=headPerp/Sqrt[N[headPerp.headPerp]];
        {Line[{a,b-abUnit*headLength}],
        Polygon[{b,b-abUnit*headLength+headPerp*headWidth/2*headLength,
            b-abUnit*headLength,
            b-abUnit*headLength-headPerp*headWidth/2*headLength}]}];
```

Veamos un ejemplo

```
g = Graphics3D[{
                    Ejes3D[-1, 3, -1, 3, -1, 3],
                    Arrow3D[P, Q],
                    Arrow3D[P, R],
                    Arrow3D[{0, 0, 0}, {1, 1, 1}]
                }, Boxed -> False,
                ViewPoint -> {2.426, 2.190, 0.878}] //Show
```

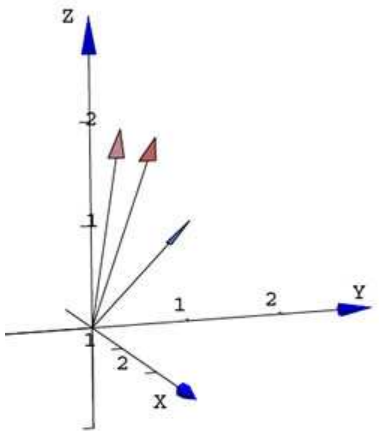

Figura 8.

Parametrización de una curva en el espacio

La forma paramétrica de una curva en 2D o 3D o incluso de una superficie, es fundamental a la hora de trazar su gráfica, como veremos.

Definición

Si $x = x(t)$, $y = y(t)$ y $z = z(t)$ son funciones continuas en un intervalo I, entonces el conjunto de tripletes ordenados $C = \{(x(t), y(t), z(t)) \mid t \in I)\}$ se denomina *curva parametrizada en el espacio tridimensional*. Las funciones $x = x(t)$, $y = y(t)$ y $z = z(t)$ se denominan *ecuaciones paramétricas* de C donde t es el parámetro.

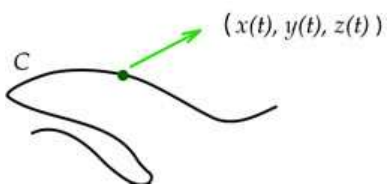

Figura 9.

Algunas parametrizaciones útiles, en dos dimensiones, son

1. Segmento de recta que une A con B: $(x, y, z) = A + t(B - A), \quad t \in [0, 1]$
2. Círculo, en el plano XY, de centro (h, k) y radio r: $(h + r\cos(t), k + r\,\text{sen}(t), 0), \quad t \in [0, 2\pi]$
3. Elipse $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$: $(h + a\cos(t), k + b\,\text{sen}(t), 0), \quad t \in [0, 2\pi]$
4. Hipérbola $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$: $(h + a\sec(t), k + b\tan(t), 0), \quad t \in [0, 2\pi]$
5. Para las curvas, en los planos XY, XZ y YZ, con ecuación funcional, se puede tomar la variable independiente como parámetro.

EJEMPLO

Grafiquemos el círculo $(x-2)^2 + (y-3)^2 = 4$ sobre el plano $z = 1$.

- Aunque podríamos usar `ParametricPlot3D[]` para graficar el círculo, vamos a usar una lista de líneas para dibujar esta curva.

  Las líneas unen los puntos *z(t)* = $(2 + r * \cos t,\ 3 + r * \text{sen}\, t,\ 1), \quad t \in [0, 2\pi]$. Los puntos son igualmente espaciados. Usaremos 30 puntos, por lo que el paso podría ser dt$= 2\pi/30$.

  Para hacer la lista de puntos usamos el comando `Table[]`

- Para dibujar el plano $z = 1$, es conveniente dibujar con `ParametricPlot3D[]` para que la circunferencia no sea parcialmente ocultada por un solo gran polígono. El paquete CurvesGraphics, que veremos más adelante, ofrece una buena solución al problema de ocultamiento de curvas sobre superficies.

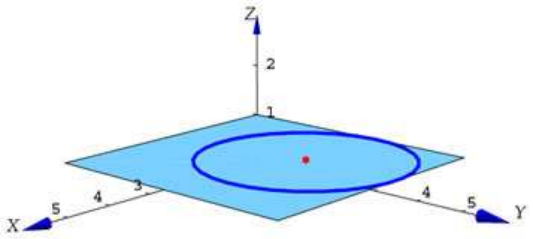

Figura 10.
[Ver en 3D- versión 1: círculo sobre un solo gran polígono]
[Ver en 3D- versión 2: con polígono construido con ParametricPlot3D]

El código es

```
r = 2;                                      (*radio*)
nptos = 30;                                 (*numero de puntos*)
dt = (2Pi -0)/nptos;                        (*paso*)
z[t_] = {2 + r*Cos[t], 3 + r*Sin[t], 1};    (*parametrizacion*)
circulo =Line[Table[z[tetha], {tetha, 0, 2Pi, dt}]];

Q = {5, 0, 1}; P = {0, 0, 1}; R = {0, 5, 1};
         (* plano P + t*(Q - P) + s*(R - P)= {5 t, 5 s, 1 *)
planoz1 = ParametricPlot3D[{5 t, 5 s, 1, EdgeForm[]},
                           {t, 0, 1}, {s, 0, 1},
                           DisplayFunction -> Identity];

g = Graphics3D[{
                 Ejes3D[-1, 6, -1, 6, -1, 3],
                 GrayLevel[0.68],
                 AbsoluteThickness[2],  (*Grosor de la linea*)
                 circulo,
                 AbsolutePointSize[4],
```

```
                RGBColor[1, 0, 0],
                Point[{2, 3, 1}]
              }, Boxed -> False,
                  ViewPoint -> {2.426, 2.190, 0.878}];

graf = Show[{g, planoz1}, DisplayFunction -> \$DisplayFunction];
JavaView[graf];
```

NOTA: en el ejemplo anterior, se puede implementar el plano como un paralelogramo,

```
Q = 5, 0, 1; P = 0,0, 1; R = 0, 5, 1; S = (Q - P) + (R - P) + P;
planoz1 =Polygon[Q, P, R, S, Q];
```

En este caso, esto no es conveniente; la visibilidad del círculo se vería afectada como se ve en la versión 1.

Discos

EJEMPLO

Vamos ahora a recortar el círculo, es decir, vamos a dibujar el disco $(x-2)^2+(y-3)^2=4$ sobre el plano $z=1$.

Lo mejor sería hacer una lista de triángulos con un vértice en el centro del círculo. Esto se hace con

```
disco = Table[Polygon[{z[t], z[t + dt], {2,3, 1}, z[t]}],
              {t, 0,2Pi,dt}];
```

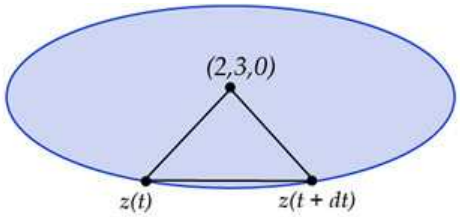

Figura 11.

Usaremos el comando `EdgeForm[]` para ocultar los bordes y el comando `SurfaceColor[]` para definir un color para esta superficie.

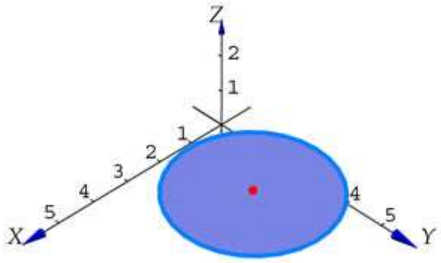

Figura 12.

El código es

```
r = 2; nptos = 30;
dt = (2Pi - 0)/nptos;
z[t_] = {2 +r*Cos[t],3 + r*Sin[t], 1};
circulo = Line[Table[z[t], {t, 0, 2Pi,dt }]];

disco = Table[
                Polygon[{z[t], z[ + dt],
                        {2, 3, 1},
                        z[t]}],
                        {t, 0, 2Pi,dt }
            ];

g = Graphics3D[{
```

```
                Ejes3D[-1, 6, -1, 6, -1, 3],
                GrayLevel[0.68],
                AbsoluteThickness[2],  (*Grosor de la linea*)
                circulo,
                EdgeForm[],
                SurfaceColor[RGBColor[0.701961, 0.701961, 1]],
                disco,
                AbsolutePointSize[4],
                RGBColor[1, 0, 0],
                Point[{2, 3, 1}]
                }, Boxed -> False,
                   ViewPoint -> {1.774, 1.815, 2.238}
              ];

Show[g];
```

Superficies regladas

Una gran cantidad de superficies son generadas por una familia de rectas, a este tipo de superficies se les llama superficies regladas. Dentro de este tipo se superficies se encuentran las superficies cónicas y los cilíndricas.

Definición

Un cilindro es el conjunto de todos los puntos que están sobre todas las rectas paralelas a una recta L dada las cuales pasan sobre una curva C que se encuentra sobre un plano P. A la recta L se le llama generatriz y a la curva C directriz.

La figura 13. ilustra la idea que esta detrás de una superficie cilíndrica, en este caso la curva directriz esta sobre el plano $z = 0$ y la generatriz es una recta paralela al eje z.

Figura 13.

Note que si queremos trazar una porción de una superficie cilíndrica, básicamente lo que debemos hacer es construir una serie de polígonos entre las dos curvas que delimitan dicha porción.

EJEMPLO

Consideremos el problema de dibujar la porción de la superficie $z = 4 - x^2$ que se encuentra en el primer octante comprendida entre los planos $y = x$ y $x + y = 5$.

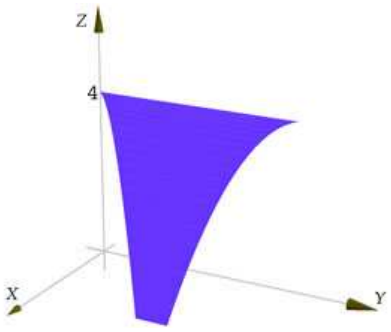

Figura 14
[Ver en 3D - JavaView]

En este caso la superficie cilíndrica que queremos dibujar es una porción de la superficie $z = 4 - x^2$. Para esto encontramos las curvas directrices

- $z = 4 - x^2 \cap y = x \Rightarrow r_1(t) = (t, t, 4 - t^2)$.
- $z = 4 - x^2 \cap x + y = 5 \Rightarrow r_2(t) = (t, 5 - t, 4 - t^2)$.

y construimos una serie de polígonos que unen dichas curvas. El código necesario para dibujar la superficies es el siguiente

```
(* Curvas directrices *)

r1[t_] := {t, t, 4 - t^2};

r2[t_] := {t, 5 - t, 4 - t^2};

(* Se generan los poligonos, estos van de r1 hasta r2 *)

dt = 0.1;

polsup = Table[
      Polygon[
        {
          r1[t],
          r2[t],
          r2[t + dt],
          r1[t + dt],
          r1[t]
          }
        ],
      {t, 0, 2, dt}
      ];

(* se construye la grafica *)

sup = Graphics3D[{
                  Ejes3D[0.5, 4, -0.5, 7, -0.5, 6],
                  {EdgeForm[], polsup}
                 },Boxed -> False ];

Show[sup, ViewPoint -> {2.805, 1.489, 1.168}];
```

Ahora, podemos usar esta idea para dibujar un sólido cuyas fronteras son superficies cilíndricas, por ejemplo, el sólido del primer octante limitado por las superficies

- $(x-2)^2 + (y-2)^2 = 1$
- $x = y = 2$
- $x + y + z = 4$

se muestra en la figura

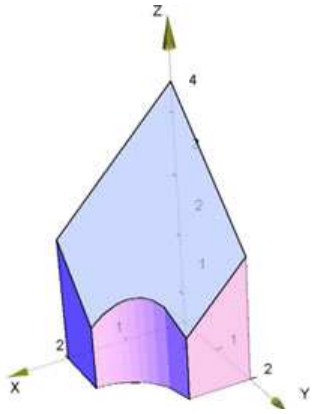

Figura 15
[Ver en 3D - JavaView]

Superficies cónicas

Definición (superficie cónica)

Es el conjunto de todos los puntos que están sobre todas las rectas que pasan por una punto fijo llamado vértice y se apoyan sobre una curva directriz.

La figura 1 ilustra la definición de una superficie cónica, en este caso el vértice es un punto sobre el eje z y la directriz es una círculo sobre el plano $z = 0$ (únicamente se ha dibujado una rama del cono).

Figura 16

EJEMPLO

Dibuje el cono con vértice en en punto $(0, 0, 5)$ y directriz $x^2 + y^2 = 4$.

En este caso construimos un conjunto de poligonos (triángulos) con dos se sus vértices sobre el círculo $x^2 + y^2 = 4$ y el otro sobre el vértice del cono $(0, 0, 5)$. El código necesario para generar el cono es el siguiente y su gráfica se muestra en la figura

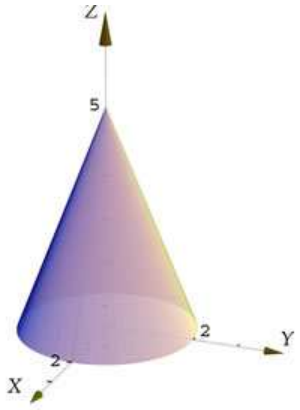

Figura 17

```
dt = 0.1;
vertice = {0, 0, 5};

r[t_] := {2 Cos[t], 2 Sin[t], 0};

cono = Table[
            Polygon[{
                     r[t],
                     r[t + dt],
                     vertice,
                     r[t]
                    }],
      {t, 0, 2 pi, dt}
      ];

sup = Graphics3D[{Ejes3D[-1, 6, -1, 6, -1, 3],
```

```
            EdgeForm[],
            cono
          },
Boxed -> False ];
```

Sector de una Superficie

Vamos a describir como se puede implementar un trozo de superficie (que no es necesariamente generada por una curva) usando su proyección. La idea es observar una manera de implementar un gráfico con el próposito de usar métodos similares para otros tipos de superficies que no son accesibles directamente con las herramientas implementadas en *Mathematica* .

EJEMPLO

Consideremos el problema de dibujar el sólido Q limitado por el paraboloide $z = x^2 + y^2$ el plano $z - y = 6$, en el primer octante.

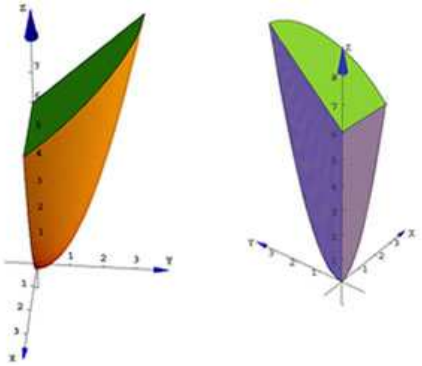

Figura 18

Para implementar este gráfico debemos implementar cada una de sus partes por separado

1. Paraboloide.

   En el caso de este sector de paraboloide, una idea para su implementación es hacer una especie de abanico con vértice en la parte más alta, como se muestra en la figura que sigue.

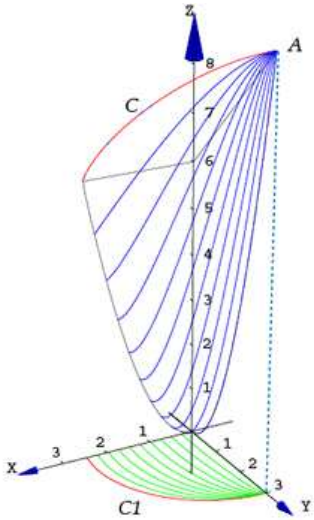

Figura 19.

   La curva C se proyecta en la curva $C1$; las trazas que conforman el abanico son una deformación de la curva C y por tanto, las curvas de nivel correspondientes son una deformación de la curva $C1$

   Bien, vamos a hacer el tejido de polígonos en la región de proyección del trozo de paraboloide, en el plano XY, y luego proyectar estos polígonos al paraboloide.

   Para hacer este tejido, vamos a tomar la curva $C1$ y deformarla, multiplicando su coordenada x por un parámetro que

toma valores entre 0 y 1. Esto produce un conjunto de curvas, todas con el mismo número de puntos. Luego, formamos los polígonos de la telaraña uniendo cada par de puntos de una curva C_i con el respectivo par de puntos en la curva siguiente C_{i+1}. Luego esta 'telaraña' la proyectaremos al paraboloide.

La curva $C1$ de intersección entre el plano $z - y = 6$ con el paraboloide $z = x^2 + y^2$, proyectada en el plano XY, tiene ecuación $x^2 + (y - 1/2)^2 = 25/4$, o sea, un círculo de radio $5/2$. Su parametrización (en el primer octante) en el plano XY es

$$C1: \quad \left(\frac{5}{2}\cos t, \frac{1}{2} + \frac{5}{2}\operatorname{sen} t, 0\right), \quad t \in \left[-0.201358,\ \frac{\pi}{2}\right]$$

Para obtener una familia de curvas a partir de $C1$, usamos la función

```
f[ra_, t_, s_] = {s*ra*Cos[t], 1/2 + ra*Sin[t], 0}, (* s en [0,1] *)
```

y la familia de curvas C_i se genera con

```
f[ra_, t_, s_] = {s*ra*Cos[t], 1/2 + ra*Sin[t], 0}, (* s en [0,1] *)

angIni = -0.201358;
    dt = (-angIni + Pi/2)/50;      (*de angIni a Pi/2 en 50 pasos*)
curvas = Table[f[5/2, i, j], {j, 0, 1, 0.1}, {i, angIni, Pi/2, dt}];
          (*dt no ajusta bien, asi que agregamos la ultima curva *)
AppendTo[curvas, Table[{5/2*Cos[t], 1/2 + 5/2*Sin[t], 0}, {t, angIni, Pi/2, dt}]];
```

Las curvas y la 'telaraña' se muestran en las siguientes figuras, junto con el código necesario

```
(*numero curvas nc*)
 nc = Length[curvas];
        (*todas las curvas tienen np puntos*)
 np = Length[curvas[[1]]];
 curvaC = Line[curvas[[nc]]];

        (*curvas ci*)
 netC = Table[Line[curvas[[ci]]], {ci, 1, nc - 1}];

        (*  cada poligono esta formado por dos puntos en la curva ci
            curvas[[ci, pi]], curvas[[ci, pi+1]]
            y dos puntos en la curva ci+1
            curvas[[ci + 1, pi]], curvas[[ci + 1, pi + 1]], *)

 netP = Table[Polygon[{curvas[[ci, pi]],
                       curvas[[ci + 1, pi]],
                       curvas[[ci + 1, pi + 1]],
                       curvas[[ci, pi + 1]],
                       curvas[[ci, pi]]
                      }], {pi, 1, np - 1}, {ci, 1,nc - 1}];

   g = Graphics3D[{ Ejes3D[-1, 4, -1, 4, -0.1, 3],
                    netP,
                    RGBColor[1, 0, 0],
                    AbsoluteThickness[2],
                    curvaC
                   }, Boxed -> False,
                      ViewPoint -> {2.580, 1.056, 1.918}
                  ];
       Show[g];
```

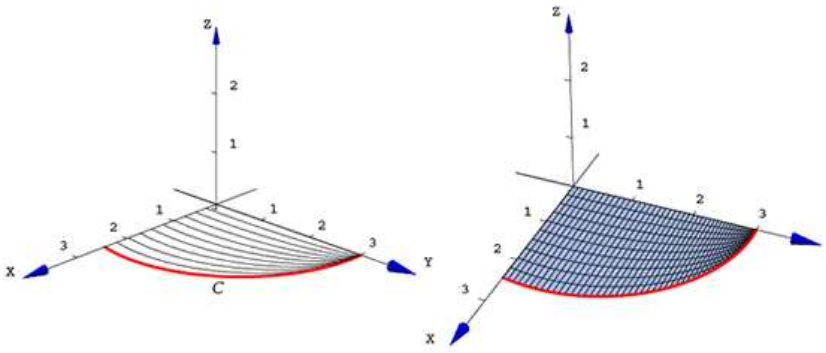

Figura 19.

Luego, cada polígono es proyectado sobre el paraboloide, para hacer esto, redefinimos la función `f[ra_, t_, s_]`

```
f[ra_, t_, s_] = {s*ra*Cos[t],1/2 + ra*Sin[t], (s*ra*Cos[t])^2 + (1/2 + ra*Sin[t])^2};
corteParaboloide = Table[Polygon[{     curvas[[ci, pi]],
                                       curvas[[ci + 1, pi]],
                                       curvas[[ci + 1, pi + 1]],
                                       curvas[[ci, pi + 1]],
                                       curvas[[ci, pi]]
                                 }], {pi, 1, np - 1}, {ci, 1,nc - 1}];

  g = Graphics3D[{ Ejes3D[-1, 4, -1, 4, -1, 9],
                   SurfaceColor[RGBColor[1, 1, 0]],
                   corteParaboloide
                   }, Boxed -> False,
                      ViewPoint -> {2.493, 0.673, 2.186}
                ];
      Show[g];
```

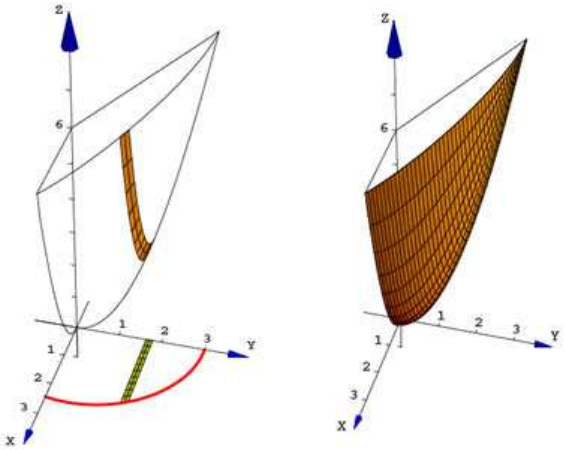

Figura 20.

2. El plano $z + y = 6$

Ya que en este caso el plano comparte la proyección sobre, el plano XY, con el paraboloide, podemos aprovechar la construcción del paraboloide para construir el plano. Nada más se debe modificar la coordenada 'z' de la función f[ra_, t_, s_]. Agregamos el siguiente código

```
f[ra_, t_, s_] = {s *ra*Cos[t], 1/2 + ra*Sin[t],  6 + 1/2 + ra*Sin[t]};
curvas = Table[f[5/2, i, j], {j, 0, 1, 0.1}, {i, angIni, Pi/2, dt}];
AppendTo[curvas,
          Table[{5/2*Cos[t],
                 1/2 + 5/2*Sin[t],
                (5/2*Cos[t])^2 + (1/2 + 5/2*Sin[t])^2},
                {t, angIni, Pi/2, dt}]];
 planoT = Table[Polygon[{curvas[[ci, pi]],
                         curvas[[ci + 1, pi]],
```

```
                        curvas[[ci + 1, pi + 1]],
                        curvas[[ci, pi + 1]],
                        curvas[[ci, pi]]
                        }], {pi, 1, np - 1}, {ci, 1,nc - 1}];

g = Graphics3D[{ Ejes3D[-1, 4, -1, 4, -1, 9],
                 SurfaceColor[RGBColor[1, 1, 0]],
                 corteParaboloide,
                 SurfaceColor[RGBColor[0.501961, 1, 0]],
                 planoT
               }, Boxed -> False,
                  ViewPoint -> {2.493, 0.673, 2.186}
      ];

 Show[g];
```

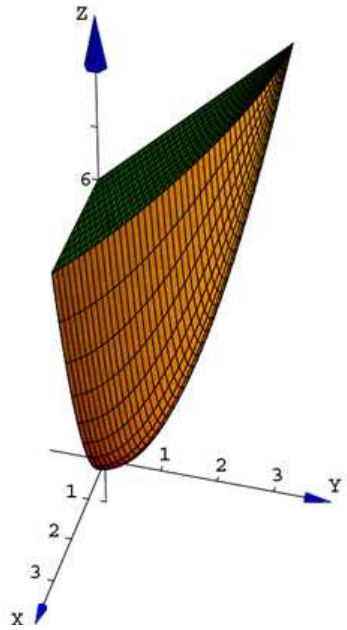

Figura 21.

3. Los planos laterales,en los planos XZ y YZ, son más sencillos. En el plano XZ, el plano va de la curva $z = x^2$ hasta la curva $z = 6$ y en el plano YZ el plano va de la curva $z = y^2$ hasta la curva $z = y + 6$. Simplemente hacemos una grupo de polígonos desde la curva de abajo hasta la curva de arriba. En el código que se agrega, se incluyen los bordes (curvas) del sólido.

```
 (*-------------bordes ----------------*)
   cxz = Line[Table[{t, 0, t^2}, {t, 0, Sqrt[6], 0.1}]];
   cyz = Line[Table[{0, t, t^2}, {t, 0, 3, 0.1}]];
   cpl1 =Line[Table[{5/2*Cos[t],
                     1/2 + 5/2*Sin[t],
                     (5/2*Cos[t])^2 + (1/2 + 5/2*Sin[t])^2}
                   ,{t,angIni, Pi/2, dt}]];
   cpl2 = Line[{{Sqrt[6], 0, 6}, {0, 0, 6}, {0, 3, 9}}];
(*-------------paredes-----------------*)
   c2[y1_] = {0, y1, 6 + y1};
   c1[y1_] = {0, y1, y1^2};
   di = 0.1;
   paredyz =Table[Polygon[{c1[i], c1[i + di],
                           c2[i + di], c2[i], c1[i]}]
                  ,{i, 0,3 - di, di}];

   c2[x1_] = {x1, 0, 6};
   c1[x1_] = {x1, 0, x1^2};
   paredxz =Table[Polygon[{c1[i], c1[i + di], c2[i + di], c2[i], c1[i]}]
                   ,{i, 0,Sqrt[6], di}];


 g = Graphics3D[{ Ejes3D[-1, 4, -1, 4, -1, 9],
                  SurfaceColor[RGBColor[1, 1, 0]],
                  corteParaboloide,
                  SurfaceColor[RGBColor[0.501961, 1, 0]],
                  planoT,
```

```
                        SurfaceColor[RGBColor[0.501961, 0.501961, 1]],
                        paredyz, paredxz,
                        cxz, cyz, cplano1, cplano2
                      }, Boxed -> False,
                         ViewPoint -> {2.493, 0.673, 2.186}
                ];
Show[g];
```

4. Como un ejemplo curioso, presentamos una implementación de la botella de Klein [15]

Figura 22.

Fue generada con este código [15]

```
color = RGBColor[0, 0, 0.627451];
bot = {(2.5 + 1.5 Cos[v]) Cos[u], (2.5 + 1.5 Cos[v]) Sin[u], -2.5 Sin[v], EdgeForm[]};


mid = {(2.5 + 1.5 Cos[v]) Cos[u], (2.5 + 1.5 Cos[v]) Sin[u], 3v, EdgeForm[]};


han = {2 - 2 Cos[v] + Sin[u], Cos[u], 3v, EdgeForm[]};


top = {2 + (2 + Cos[u]) Cos[v], Sin[u], 3Pi + (2 + Cos[u]) Sin[v], EdgeForm[]};


bottom = ParametricPlot3D[bot, {u, 0, 2Pi}, {v, 0, Pi}, PlotPoints -> {32,
16}                        ,AmbientLight -> color, Boxed -> False, Axes -> False];
middle = ParametricPlot3D[mid, {u, 0, 2Pi}, {v, 0, Pi}, PlotPoints -> {32, 16},
                          AmbientLight -> color, Boxed -> False, Axes -> False];
topper = ParametricPlot3D[top, {u, 0, 2Pi}, {v, 0, Pi}, PlotPoints -> {32,
16}                        ,AmbientLight -> color, Boxed -> False, Axes -> False];
handle = ParametricPlot3D[han, {u, 0, 2Pi}, {v, 0, Pi},
                          PlotPoints -> {32, 16}, AmbientLight -> color,
                          Boxed -> False, Axes -> False];
todo = Show[{handle, topper, middle, bottom}, Boxed -> False,
             Axes -> False, AmbientLight -> color]
```

Implementación de superficies usando DrawGraphics y CurvesGraphics

En las secciones anteriores se vieron métodos generales para implementar superficies de cualquier tipo, usando mayormente primitivas. DrawGraphics y CurvesGraphics ofrecen nuevas herramientas para los casos generales y también para casos particulares.

DrawGraphics es un paradigma alternativo para producir gráficos en *Mathematica*. Las rutinas básicas de DrawGraphics producen los gráficos de *Mathematica*, sin desplegarlos, y extraen las primitivas del objeto producido por *Mathematica* para poder combinar diferentes elementos gráficos en un solo contexto.

DrawGraphics y CurvesGraphics son paquetes que extraen las primitivas de los gráficos y los combinan para producir gráficos con un código más natural. CurvesGraphics funciona usando DrawGraphics y extiende los comandos para curvas orientadas.

Instalación

Se Podría copiar la carpeta `DrawGraphics` y el archivo `CurvesGraphics.m` en el directorio

```
C:\...\Wolfram Research\Mathematica\5.0\AddOns\Applications
```

Para llamar a ambos paquetes se ejecuta el código

```
Needs["DrawGraphics`DrawingMaster`"];

Needs["CurvesGraphics`"];
```

Comandos básicos

Los nombres de los comandos en DrawGraphics, son similares a los de *Mathematica*, solo se agrega 'Draw'. Por ejemplo

Mathematica		DrawGraphics
ParametricPlot3D		ParametricDraw3D
Plot3D		Draw3D
Show		Draw3DItems
InequalityPlot3D		InequalitDraw3D

Las rutinas 'Draw' tienen la misma forma y aceptan las mismas opciones que las rutinas 'Plot' excepto las opciones AspectRatio, Frame, PlotLabel, etc. que no producen ningún efecto.

Algunos comandos de *Mathematica* no están presentes en los paquetes, pero se pueden incorporar como se verá más adelante. También hay comandos nuevos en estos paquetes.

A través de los ejemplos que siguen, vamos a mostrar algunas de las calidades de este nuevo paradigma, aplicándolo a los ejemplos anteriores y a otros ejemplos nuevos

La documentación de DrawGraphics viene incluida en la carpeta DrawGraphics, ([12]) . Un manual actualizado de CurvesGraphics se puede obtener en ([12])

Ejemplos

1. Ejes 3D

   Podemos hacer un comando que dibuje un sistema de ejes 3D usando el comando ArrowLine3D de DrawGraphics. También podemos usar Arrow3D para aplicarlo a curvas. La sintaxis es

   ```
   Arrow3D[base,cabeza, opciones]

   ArrowLine3D[ puntos,opciones...]
   ```

   ArrowLine3D recibe una lista puntos = {pt1, pt2,...,ptk}

   La opción 'HeadSegments3D' especifica el número de segmentos triángulares usados para dibujar el cono de la flecha. El default es 10. Una flecha plana se logra con 2.

   ```
   flechas1 = Draw3DItems[{Arrow3D[{0, 0, 0}, {0, 1, 1}],
                           RGBColor[0, 0, 1],
                           Arrow3D[{0, 0, 0}, {1, 0, 1}],
                           RGBColor[0.5, 0, 1],
                           AbsoluteThickness[2],
                           SurfaceColor[RGBColor[0, 1, 0]],
                           Arrow3D[{0, 0, 0}, {1, 1, 0}]
                           },
                           ViewPoint -> {2.209, 2.115, 1.449},
                           Boxed -> False,
                           PlotRange -> All];
   ```

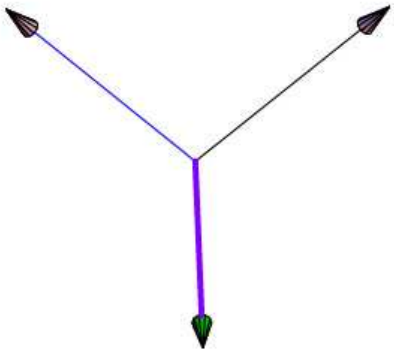

Figura 22.

```
SetOptions[Arrow3D, HeadSegments3D -> 2];
flechas2 = Draw3DItems[{Arrow3D[{0, 0, 0}, {0, 1, 1}],
                        RGBColor[0, 0, 1],
                        Arrow3D[{0, 0, 0}, {1, 0, 1}]
                       },
                       ViewPoint -> {2.209, 2.115, 1.449},
                       Boxed -> False,
                       PlotRange -> All];
```

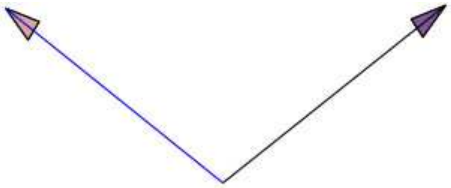

Figura 23.

2. **Ejes 3D**

Usamos un código similar al visto más arriba

```
Ejes[xmin_, xmax_, ymin_, ymax_, zmin_, zmax_] := {
               (*Flechas*)
                SurfaceColor[RGBColor[0, 0, 1]],
                ArrowLine3D[{{xmin, 0, 0}, {0, 0, 0}, {xmax, 0, 0}}],
                ArrowLine3D[{{0, ymin, 0}, {0, 0, 0}, {0, ymax, 0}}],
                ArrowLine3D[{{0, 0, zmin}, {0, 0, 0}, {0, 0, zmax}}],
                GrayLevel[0],
                Text["X", {xmax, -0.3, 0}],
                Text["Y", {-0.3, ymax, 0.1}],
                Text["Z", {0, -0.2, zmax}],
                (* numeros *)
                Table[Text[i, {i, -0.3, 0}], {i, 1, xmax - 2, 1}],
                Table[Text[i, {-0.3, i, 0}], {i, 1, ymax - 2, 1}],
                Table[Text[i, {-0.3, -0.3, i}], {i, 1, zmax - 2, 1}],
                RGBColor[0, 0, 0.627451],
                (* rayitas *)
                Table[Line[{{-0.1, i, 0}, {0, i, 0}}], {i, 1, ymax - 1, 1}],
                Table[Line[{{i, -0.1, 0}, {i, 0, 0}}], {i, 1, xmax - 1, 1}],
                Table[Line[{{0, -0.1, i}, {0, 0, i}}], {i, 1, zmax - 1, 1}]
               };

               SetOptions[Arrow3D, HeadSegments3D -> 10];
               elGraf = Draw3DItems[{
                                     GrayLevel[0.501961],
                                     Ejes[-0.5, 3, -0.5, 3, -0.5, 5]},
                                     ViewPoint -> {3.221, 0.246, 1.008
                                    },
                                     Boxed -> False,
                                     PlotRange -> All];
```

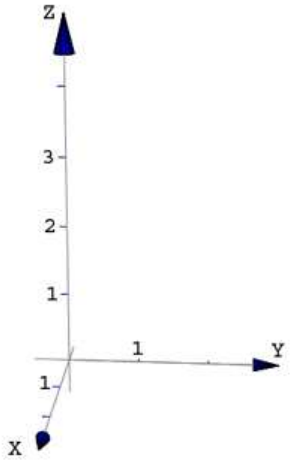

Figura 24.

3. **Usando JavaView**

Procedemos como se indicó más arriba.

Primero ejecutamos el código

```
Needs["JLink`"];
InstallJava[];
<< JavaView`JLink`
viewer = InstallJavaView[];
```

y luego ya podemos levantar la ventana de JavaView

```
JavaView[elGraf];
```

4. **'RealTime3D'**

'RealTime3D' nos permite interactuar con el gráfico directamente en el 'notebook'. Se ejecuta el código (para usar 'RealTime3D' se debió eliminar StyleForm[] de Ejes[...])

```
<< RealTime3D`
Show[elGraf];
<< Default3D`
```

5. **Draw3D**

Veamos un ejemplo de cómo usar Draw3DItems y Draw3D

```
plot1 = Draw3DItems[{
                          Ejes2[-0.5, 3, -0.5, 3, -0.5, 9],
                          Draw3D[x^2 + y^2, {x, 0, 3}, {y, 0, 3},
                                 PlotPoints -> 20] //UseWireFrame,
                          Draw3D[y + 6, {x, -1, 3}, {y, -1, 3}]
                          },
                          Boxed -> False,
                          ViewPoint -> {1.324, -2.001, 2.386},
                          ImageSize -> 100];
```

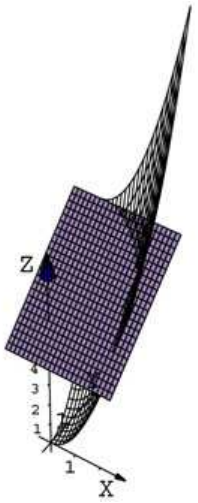

Figura 25.

6. **ParametricDraw3D**

Veamos un ejemplo de cómo usar ParametricDraw3D

```
plot2 = Draw3DItems[{EdgeForm[],
                             Ejes2[-0.5, 3, -0.5, 3, -0.5, 9],
                             SurfaceColor[RGBColor[0, 1, 0]],
                             ParametricDraw3D[{r*Cos[t], r*Sin[t], r^2},
                                              {r, 0, 3}, {t,0, Pi/2}]
                             },
                             ViewPoint -> {0.636, -2.991, 1.449},
                             Boxed -> False
                            ];
```

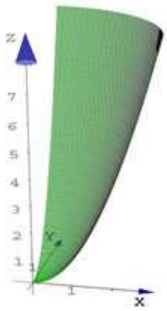

Figura 26.

7. **LinearClipPlot3D**

LinearClipPlot3D es un comando de 'CurvesGraphics' que dibuja el recorte de una superficies usando planos vía *desigualdades lineales*

```
LinearClipPlot3D[3DSurface, LinearInequality, variables]
```

Por ejemplo, observemos todos los gráficos que produce el siguiente código

```
paraboloid = ParametricPlot3D[r{Cos[t], Sin[t], r}, {r, 0, 3}, {t, 0, 2*Pi}];
GInt1 = LinearClipPlot3D[paraboloid, z <= y + x + 1  , x, y, z];

(*Como GInt1 es Graphics3D[{primitivas}] entonces  GInt1[[1]] = {primitivas} *)

intersecciones = Graphics3D[{Join[{EdgeForm[]}, GInt1[[1]]]}];
Show[intersecciones]
```

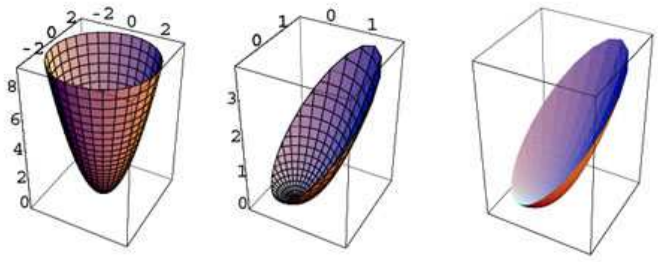

Figura 27.

8. **Sólido usando LinearClipPlot3D**

Vamos a usar LinearClipPlot3D para ayudarnos a implementar el sólido limitado por $z = x^2 + y^2$ y los planos $z = y + 6$ y $z = 6$, en el primer octante. Observe que este comando solo funciona con *desigualdades lineales*.

```
paraboloid2 =ParametricPlot3D[r{Cos[t], Sin[t], r},
                              {r, 0, 3}, {t, 0, Pi/2},
                              PlotPoints -> 40];

                ejes = Graphics3D[Ejes2[-0.5, 3, -0.5, 3, -0.5, 9]];
                GInt1 = LinearClipPlot3D[paraboloid2, 6 <= z <= y + 6  , x, y, z];
               (*GInt1 es Graphics3D[{primitivas}] entonces GInt1[[1]] = {primitivas} *)
               intersecciones = Graphics3D[{Join[{EdgeForm[]}, GInt1[[1]]]}];

                (*---------------- plano 1 ---------------------------------*)
                z2[y_] = {Sqrt[6 + y - y^2], y, 6 + y};
                z1[y_] = {0, y, 6 + y};
                dx = 0.1;
                Rplano1 = Table[Polygon[{z1[i], z1[i + dx], z2[i + dx], z2[i], z1[i]
                        }], {i, 0, 2.7, 0.1}];
                dx = 0.01;
                Rplano2 = Table[Polygon[{z1[i], z1[i + dx], z2[i + dx], z2[i], z1[i]
                        }], {i, 2.7, 3 - 0.01, 0.01}];
                (*---------------- plano 2 ---------------------------------*)
                dt = (Pi/2)/30;

                z6[te_] = {Sqrt[6]Cos[te], Sqrt[6]*Sin[te], 6 };
                Rplanoz6 = Table[
                    Polygon[{
                             z6[i], z6[i + dt], {0, 0, 6}, z6[i]
                            }],
                    {i, 0, Pi/2 - dt, dt}];

                planos = Graphics3D[{
                        {EdgeForm[],
                        SurfaceColor[RGBColor[0, 1, 0]],
                        Rplano1, Rplano2,
                        SurfaceColor[RGBColor[1, 1, 0.501961]],
                        Rplanoz6}

                        }];

                plot4 = Show[{
                               ejes, planos,  intersecciones
                             }, ViewPoint -> {2.894, -0.892, 1.510},
                                Boxed -> False,
                                PlotRange -> All]
```

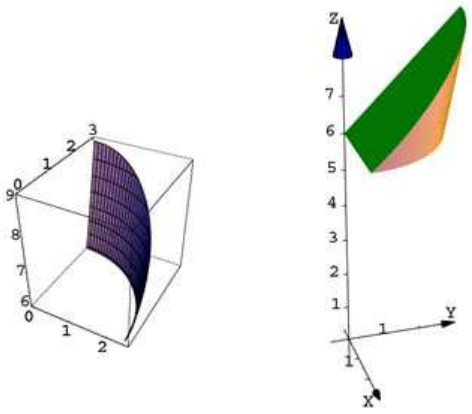

Figura 28.
[Ver en 3D con JavaView]

9. **InequalityDraw3D**

Esta rutina es muy conveniente para dibujar sectores de superficies y sólidos. Vamos a ver algunos ejemplos de su uso.

a. Sector del plano $z + y = 6$ al ser cortado por la superficie $z = 4 - x^2$ con $x \leq 2$, en el primer octante.

La idea es definir la región de la proyección del plano, en XY, con una desigualdad. InequalityDraw3D toma esta región, definida por una desigualdad, y la proyecta sobre el plano que también debe ser definido por otra desigualdad. El código generado por InequalityDraw3D incluye los polígonos de la proyección y el sector del plano. Para eliminar la proyección del gráfico, extraemos los polígonos del sector del plano

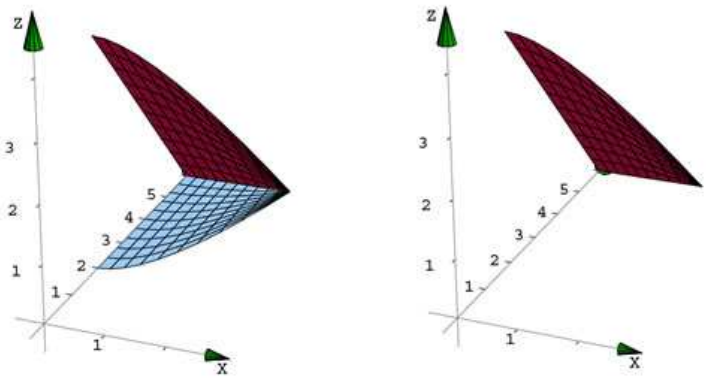

Figura 29.

En el siguiente código se muestra el procedimiento

```
 (*región de proyección en XY:  y >= 2 + x^2, {x,0, 2}, {y, 2, 6} *)

 eqns3 = y >= 2 + x^2  ∧  y ≤ 6 - z;
(regionP = InequalityDraw3D[ eqns3, {x,0, 2}, {y, 2, 6}, {z, 0, 4},
                             PlotPoints -> 10];)
elGraf = Draw3DItems[
         {
          GrayLevel[0.501961],
          Ejes[-0.5, 3, -0.5, 7, -0.5, 5],
          regionP[[1,1]] (*sin los polígonos de la proyección*)
         },
         ViewPoint -> {1.125, 2.903, 1.325},
         Boxed -> False,
         PlotRange -> All
                   ];
```

b. Sólido limitado por los planos $y = 2$, $z + y = 6$ y la superficie $z = 4 - x^2$, en el primer octante.

En este caso si nos interesa la proyección pues es parte del sólido. En la siguiente implementación se omite la pared en YZ para efectos de visualizar el interior.

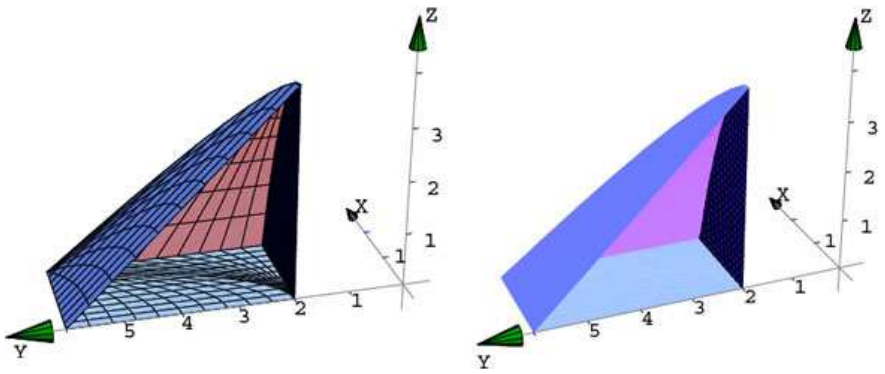

Figura 30.

En el siguiente código se nuestra el procedimiento

```
eqns2 = z ≤ 4 - x^2 ∧ y ≤ 6 - z;
(region=InequalityDraw3D[eqns2, {x, 0, 2}, {y, 2, 6}, {z, 0, 4}, PlotPoints -> 10];)

(*agregamos EdgeForm[] para ocultar la malla*)
region = Join[{EdgeForm[]}, region];


(*la pared en y=2 va de z=0 hasta z=4 - x^2 *)
zt[x_, vz_] = If[vz <= 4 - x^2, vz, 4 - x^2];
elGraf2 = Draw3DItems[
          {
           GrayLevel[0.501961],
           Ejes[-0.5, 3, -0.5, 7, -0.5, 5],
           region,
           ParametricDraw3D[{x, 2, zt[x, vz]}, {x, 0, 2}, {vz, 0, 4}]
          },
          ViewPoint -> {-2.815, 1.118, 1.510},
          AmbientLight -> RGBColor[0, 0, 0.627451],
          Boxed -> False,
          PlotRange -> All
                  ];
```

c. Sólido que corresponde a la región interior de la esfera $x^2 + y^2 + z^2 = 4$ y al interior del cilindro

$(x-1)^2 + y^2 = 1$, en el primer octante.

En este caso la proyeción sobre XY es la proyección del cilindro

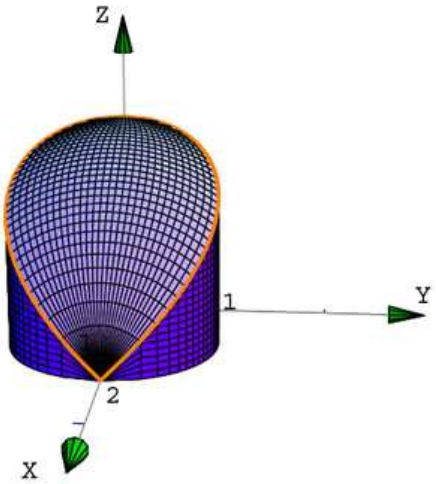

Figura 31.

[Ver en 3D con JavaView]

En el siguiente código se muestra el procedimiento

```
(*Curva intersección : Cint[x_] = {x, ±Sqrt[1 - (x - 1)^2], Sqrt[4 - 2 x] *)
(* cilindro = {x, ±Sqrt[1 - (x - 1)^2], cz[x, z]} *)

cz[x_, tz_] = If[tz ≤ Sqrt[4 - 2 x], tz, Sqrt[4 - 2 x]];
cil1[x_, tz_] = {x, Sqrt[1 - (x - 1)^2], cz[x, tz]};
cil2[x_, tz_] = {x, -Sqrt[1 - (x - 1)^2], cz[x, tz]};
Cint1[x_] = {x, Sqrt[1 - (x - 1)^2], Sqrt[4 - 2 x]};
Cint2[x_] = {x, -Sqrt[1 - (x - 1)^2], Sqrt[4 - 2 x]};

eqns1 = (x - 1)^2 + y^2 ≤ 1 ∧ x^2 + y^2 + z^2 ≤ 4;
regionEsfera = InequalityDraw3D[ eqns1, {x, 0, 2}, {y, -1, 1}, {z, 0, 2},
                                 PlotPoints -> 30];

elGraf = Draw3DItems[
         {
          GrayLevel[0.501961],
          Ejes[-0.5, 4, -0.5, 3, -0.5, 3],
          regionEsfera,
                   (* cilindro, de z=0 hasta la curva de intersección*)
          SurfaceColor[RGBColor[0.501961, 0, 1]],
          ParametricDraw3D[cil1[x, z], {x, 0, 2}, {z, 0, 2}],
          ParametricDraw3D[cil2[x, z], {x, 0, 2}, {z, 0, 2}],
          AbsoluteThickness[1.5],
          RGBColor[1, 0.501961, 0],
          ParametricDraw3D[Cint1[t], {t, 0, 2}],
          ParametricDraw3D[Cint2[t], {t, 0, 2}]
          },
          ViewPoint -> {3.221, 0.246, 1.008},
          Boxed -> False,
          AmbientLight -> RGBColor[0, 0, 1],
          PlotRange -> All
                 ];
```

d. De nuevo el Sólido Q limitado por el paraboloide $z = x^2 + y^2$ el plano $z - y = 6$, en el primer octante.

Compare con el sólido en la sección anterior

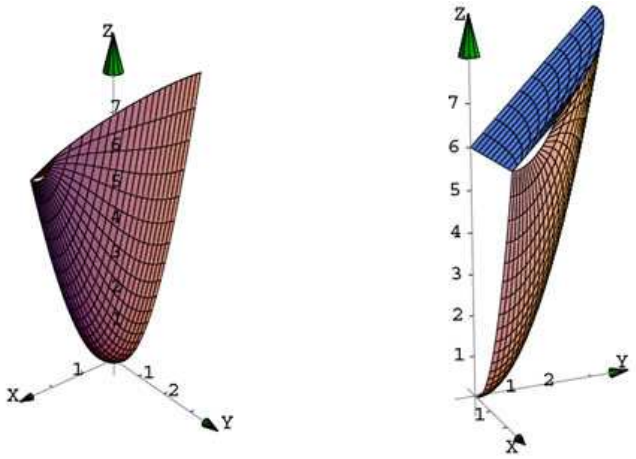

Figura 32.

Para la implementación, de nuevo, proyectamos sobre XY. El código es como sigue

```
eqns4 = x^2 + (y - 1/2)^2 ≤ 25/4 ∧ z ≤ x^2 + y^2 ∧ z ≤ 6 + y ;
regionParaboloide=InequalityDraw3D[eqns4, {x, 0, Sqrt[6]}, {y, 0, 3}, {z, 0, 9},
                                   PlotPoints -> 20];

eqns5 = x^2 + (y - 1/2)^2 ≤ 25/4 ∧ z ≤ 6 + y ;
(regionPlano = InequalityDraw3D[ eqns5, {x, 0, Sqrt[6]}, {y, 0, 3}, {z, 0, 9},
                                 PlotPoints -> 10];)

elGraf = Draw3DItems[
```

```
{
 GrayLevel[0.501961],
 Ejes[-0.5, 3, -0.5, 4, -0.5, 9],
 regionParaboloide[[1, 1]],
 regionPlano[[1, 1]]
},
ViewPoint -> {2.962, -1.337, 0.943},
Boxed -> False,
PlotRange -> All
];
```

---

e. Disco $(x-2)^2+(y-3)^2=4$ sobre el plano $z=1$.

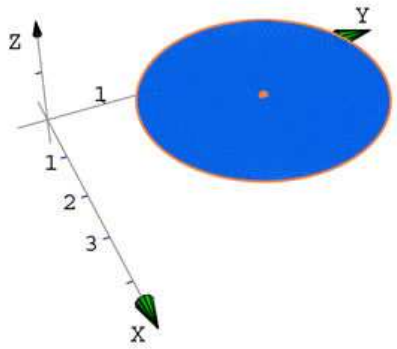

Figura 33.

El código es como sigue

```
eqns6 = (x - 2)^2 + (y - 3)^2 ≤ 4 ∧ z <= 1 ;
regionC = InequalityDraw3D[ eqns6, {x, 0, 4}, {y, 1, 5}, {z, 0, 1},
                            PlotPoints -> 35];
regionC = Join[{EdgeForm[]}, regionC[[1,1]]];

elGraf = Draw3DItems[
         {
          GrayLevel[0.501961],
          Ejes[-0.5, 5, -0.5, 6, -0.5, 2],
          SurfaceColor[RGBColor[0, 0.501961, 1]],
          regionC,
          AbsoluteThickness[1],
          RGBColor[1, 0.501961, 0.25098],
          ParametricDraw3D[{2 + 2*Cos[t], 3 + 2*Sin[t], 1}, {t, 0, 2Pi}],
          AbsolutePointSize[4],
          Point[{2, 3, 1}]
          },
          ViewPoint -> {2.354, -1.062, 2.186},
          Boxed -> False,
          PlotRange -> All
          ];
```

---

10. **PlotCurveOnSurface3D: curvas sobre una supeficie**

En *Mathematica* las curvas sobre objetos gráficos no se ven, en general, muy bien. Esto se resuelve en CurvesGraphics con PlotCurveOnSurface3D.

En el siguiente ejemplo se dibuja una curva sobre una superficie. Se habilita la orientación a la curva con la opción '`Oriented -> True`'

```
plot5 = PlotCurveOnSurface3D[
                    {{4Cos[u]Cos[v], 3Sin[u]Cos[v], 2Sin[v]},
                               {u, 0,2Pi}, {v, -Pi/2, Pi/2},
                               PlotPoints -> {51, 41}},

                    {{t*Cos[30t] - Pi/2, t*Sin[30t] + Pi/6},
```

```
                {t, 0, 1}, Oriented -> True,
                 PlotPoints -> 100},


               Ticks -> None,
               Boxed -> False,
               Axes -> False];

ejes = Graphics3D[Ejes2[-5, 6, -5, 6, -0.5, 4]];

Show[{ejes, plot5},
      ViewPoint -> {2.021, -2.077, 1.747},
      Boxed -> False,
      PlotRange -> All]
```

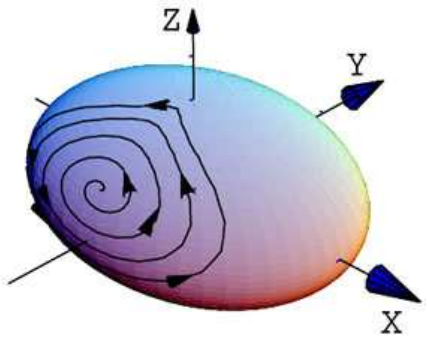

Figura 29.
[Ver en 3D con JavaView]

Bibliografía

1 Mathematica y JavaView. Consultada 24 mar. 2005. Disponible en http://www.javaview.de/mathematica/index.html

2 Martin, Kraus. LiveGraphics3D. Consultada 24 mar. 2005. Disponible en http://www.vis.informatik.uni-stuttgart.de/ kraus/LiveGraphics3D/

3 Pita R. C. 1995. Cálculo Vectorial. Prentice-Hall. México, MX, 1078 p.

4 Davis, B.1994. Calculus and Mathematica. Addison-Wesley. New York, US, 330p.

5 Polthier, K et al (Edts) 2002. Multimedia Tools for Communicating Mathemathics. Berlin, Ed. Springer-Verlag (Mathematics and Visualization). 311 p.

6 Schneider, Ph. Eberly, D. 2003. Geometric Tools for Computers Graphics. Ed. Morgan Kaufmann. San Francisco California. 1007

7 Wolfram Research. Consultado 20 de mayo 2005. Disponible en http://www.wolfram.com

8 LiveGraphics3d Home. Consultado 20 de mayo de 2005. Disponible en http://www.vis.informatik.uni-stuttgart.de/ kraus/LiveGraphics3D/

9 JavaView - Interactive 3D Geometry and Visualization. Consultado 20 de mayo 2005. Disponible en http://www.javaview.de

10 JavaView - Interactive 3D Geometry and Visualization. Consultado 20 de mayo 2005. Disponible en http://www.javaview.de

11 Download Java 2 Platform Standard Edition 1.4.2 Consultado 20 de mayo 2005. Disponible en http://java.sun.com/j2se/1.4.2/download.html

12 Arrow3D. Consultado 20 de mayo 2005. Disponible en http://library.wolfram.com/infocenter/TechNotes/4117/

13 DrawGraphics. Consultado 20 de mayo 2005. Disponible en http://home.earthlink.net/ djmp/Mathematica.html

14 CurveGraphics. Consultado 20 de mayo 2005. Disponible en http://www.dimi.uniud.it/ gorni/

15 Klein Bottle. Consultado 20 mayo 2005. Disponible en http://emsh.calarts.edu/~mathart/sw/klein/Klein.html