## Using 3D Graphics Package

The 3D Graphics Package is divided into separate Parts, one called GrafSys and one Screen3D. The former is the actual 3D transformation unit, while the latter contains the routines that draw on the screen.

Call InitGraf at the beginning of your program. This sets up the required variables and initializes the transformation and projection packages within.

The GrafSys uses ports similar to Quickdraw. The main difference is that you can have multiple GrafPort3D in a single Quickdraw GrafPort. A GrafPort3D should always reside inside a Quickdraw GrafPort. Call NewGrafPort at least once. This sets up the projection plane and initializes the projector to parallel projection. This call should be immediately followed by a call to SetEye that will define the viewangle.

After this, you usually load your objects from resource or create them with the commands NewObject, AddPoint, AddLine and AddPoly. Once an object is done constructing or loading, use the ObjRotate, ObjTranslate and ObjScale to manipulate it. Call SetEye if you want to move the camera.

**Note**: object manipulation commands (rotate and translate) fall in two different cathegories:

All the ObjRotate, ObjTranslate and ObjScale routines are independent from each other and in which sequence they are executed.

In contrast, the Obj**Free**Rotate and Obj**Free**Translate commands all depend upon their order and different orders of calling will have different results (they have a cummulative effect). Those routines were added to give you an additional degree of freedom but you should be careful if you use them since an unexperienced user cannot predict what effect a change in the sequence of commands will have.

To view an object, first call TransformObject and then DrawObject to draw it on the Screen. If you are using the ScreenObjects, use CalcScreenObject and DrawScreenObject instead.

Using the ScreenObject for your own Drawing Routines

If you have implemented your own perversely-fast graphics routines you might not want to use the in the Screen3D provided drawing routines since they rely on the normal QuickDraw routines. GrafSys provides you with an easy interface that you can use to get all the data you need to draw the object. This interface is the ScreenObject. It really is nothing else but a data structure that contains all relevant data of the *transformed* object. You can use this data to do anything that you like.

Lets have a closer look at the ScreenObject:

```
ScreenObjPtr = ^ScreenObj;

ScreenObj = record
  nhmin, nhmax, nvmin, nvmax: integer; (* new rect     *)
                               (* from last calculation *)
  hmin, hmax, vmin, vmax: integer;
     (* Rect in which ScreenObject from SECOND LAST    *)
     (* call to ClacScreenObj was drawn                *)
  Point: PointArray; (* Transformed Points of object *)
  deepz: real; (* maximum z of all Transformed Points. *)
               (* Used for Scene-Building/HL/HS Alg.   *)
  maxPoint, maxLine, maxPoly: integer;
     (* number of Points,  Lines and Polygons in this  *)
     (* Object                                         *)
  Line: LineArray;  (* Lines as defined in Parent *)
  screenx: screenPts; (* x- and y-coords of all Points *)
  screeny: screenPts; (* after transformation          *)
  Autoerase: Boolean;
  EraseType: Integer;
  screen1x: ScreenArray; (* x-coordinates for clipped *)
                         (* lines in CxxxScreenObj    *)
  screen1y: ScreenArray; (*          - " -            *)
  screen2x: screenArray; (* used in Line-Clipping mode*)
  screen2y: screenArray; (*         - " -             *)
  screenLines : Integer; (*         - " -             *)
  newLine: newLineArray; (*         - " -             *)
  Polygons: PolyArray;   (* Polygons as in Parent     *)
 end;
```

The ScreenObject contains some fields that are specific for use with the DrawScreenObject routines. However, you can use them as well in your own Programs.

nhmin, nvmin, nhmax and nvmax are four integers reserved for calculating the screen boundaries of the object to draw. CalcScreenObject and CCalcScreenObject place the information after transforming the object here.

hmin, vmin, hmax and vmax contain the objects screen boundaries from the last time the object was drawn. This is of course used by the DrawScreenObject routine to erase the old image. After drawing, DrawScreenObjet copies the contents of the nhxxx and nvxxx variables into these locations.

Point contains the coordinates of all the object points *after transformation*. You can use this information for your own depth sorting algorithms. Note that after transformation for the eye the coordinate system is moved rather than the eye. This means that the eye will always look at the XY plane.
If for example you implemented a flight simulator and moved the eye around the world, after transformation other objects distances to the eye are their distances to the global origin. This makes collision detection and distance calculation very easy.

**Warning**: If you are using the Fixed-Point version of the GrafSys, all coordinates are given in Fixed data type and you have to convert the X, Y and Z coodinates using the Fix2X call.

deepz contains the maximum (largest) Z coordinate of an object after transformation. SIce the eye (after tzransformation) is looking at the XY plane straight down the Z-achsis, use this value for queuing objects. The greater their deepz value, the further the object is from the eye. A negative values means that the whole object is behind the eye and should not be drawn if clipping is on.

maxPoint, maxLine and maxPoly  contain the number of Points, Lines and Polygons so far defined in this obejct.

screenx and screeny are two arrays that contain the screen coordinates of each transformed point.

Autoerase is a copy of the same flag used in the master object. Note that you shouldn't rely on the correctnes of this value and rather look it up in the master object itself.

EraseType contains the method of how to erase the object prior to redrawing it if Autoerase is true. Note that so far no matter what you specify the object gets erased by erasing the bounding rect.

screen1x/y and screen2x/y are four arrays that contain all screen coordinates for all lines (aka 'Line Buffer'). These coordinates are the same as in screenx/y except that this buffer is optimized for drawing:
It contains the screen coordinates of all *Lines* i.e. to draw line #5 you would

issue

```
MoveTo(screen1x[5],screen1y[5]);
DrawTo(screen2x[5],screen2y[5]);
```

As you can see, this can speed up drawing cosiderably.

**Note**: If you are using CCalcObject and clipping, those lines that completely fall offscreen will not show up in this array. Lines that are partially clipped will have their correct screen coordinates in here.

screenLines is the number of lines that are currently contained in the line buffer. Note that this number can be radically different from the number of lines defined in the object. If for example a line falls completely off the screen, the number of lines will be one less than in the objects definition.

newLine is an array that contains only boolean values. If a line begins at a new screen position and the cursor must be moved there via the MoveTo procedure, its corresponding value will be true. Otherwise you may skip the MoveTo command and simply continue drawing from the last position.

Polygons contain the polygon definitions as in master object.

To illustrate how to use the ScreenObject, Iook at how the Screen3D units DrawScreenObject command works:

```
procedure DrawScreenObject (theObject: GrafObjPtr);

  var
    index: Integer;
    r: Rect;
    x, y: integer;
    theScrnObj: ScreenObjPtr;
    thePort: Graf3DPtr;

begin
  theScrnObj := theObject^.ScreenObjLink;
                                (* get the screenObject *)
  if theScrnObj = nil then (* failsafe *)
    Exit(DrawScreenObject);

  with theScrnObj^ do
    begin
      if Autoerase then
        begin
          GetGrafPort(thePort);
          SetRect(r, theScrnObj^.hmin, theScrnObj^.vmin,
```

```
                    theScrnObj^.hmax,theScrnObj^.vmax);
            EraseRect(thePort^.viewPlane);
          end; (* if autoerase *)

(* now draw the object. Use the Line Buffer for this *)
        for index := 1 to screenLines do
          begin
            if newLine[index] then
              MoveTo(screen1x[index], screen1y[index]);
            LineTo(screen2x[index], screen2y[index]);
          end;

(* since clipping might have destroyed/rendered useless the
  min/max values, rebuild them *)
        hmax := -32000;
        hmin := 32000;
        vmax := -32000;
        vmin := 32000;

        for index := 1 to screenLines do
          begin
            x := screen1x[index];
            y := screen1y[index];
            if x > hmax then (* do bounds checking *)
              hmax := x;
            if x < hmin then
              hmin := x;
            if y > vmax then
              vmax := y;
            if y < vmin then
              vmin := y;
            x := screen2x[index];
            y := screen2y[index];
            if x > hmax then (* do bounds checking *)
              hmax := x;
            if x < hmin then
              hmin := x;
            if y > vmax then
              vmax := y;
            if y < vmin then
              vmin := y;
          end;
        hmax := hmax + 1;
        vmax := vmax + 1;
        hmin := hmin - 1;
        vmin := vmin - 1;

      end; (* with *)
  end;
```

881 versus FixedPoint Arithmetic

Response to the initial publication of the GrafSys caught me completely off-

guard. An overwhelming number of people asked me if it was possible to supply a version that uses fixed-point arithmetic instead of relying on the 881 math coprocessor.

As a result, there are now two versions of the GrafSys library. Those libraries that contain the word *'fix'* in its name work with any Macintosh. This is called 'the fixed version'. The other (original) version still requires at least a 020 processor and a math coprocessor.

Some people commented on the fact that Fixed-Point arithmetic is 'wickedly fast'. I was really astonished to see just how fast these routines were. If precision is not an issue, you might want to use the fixed version since it works with more macs.

Using the 881 Version

To use the GrafSys, include the file GrafSys.lib and GrafSys.Int into your project.

If you plan on using the provided screen drawing routines, you will also have to include the files Screen3D.lib and Screen3D.int into your project.

If you plan on writing two versions of the same program one using the 881 version, the other the fixed version, make sure you read the 'Compatability' paragraph, below.

Using FixedPoint Version

The fixed version runs on any Mac. Instead of using the math coprocessor it uses fixed point arithmetic that is lighning fast but not as accurate as real numbers. You should not use big numbers when using the fixed point version. Numbers greater than 32000 will surely produce strange results under certain conditions, numbers grater than 65000 are illegal. Note that coordinates easily can become this large if you use large values for both coordinates and translation.

To use the GrafSys, include the file GrafSys.fix.lib and GrafSys.fix.Int into your project. In addition, you must include the SANElib.lib into your project.

If you plan on using the provided screen drawing routines, you will also have to include the files Screen3D.fix.lib and Screen3D.fix.int into your project.

Compatability

The two versions of GrafSys are *Source Level* compatible. Well, almost. If you use the RealVector4 type in your programs instead of the Vector4 type you will have no compatability problems.

Object resources (the '3Dob' type) are *totally compatible*. The fixed library automatically loads and converts the floating point definitions to fixed-point while loading and back prior to writing them.

Make sure you never directly access an objects point definition since they are different in the two versions. Instead, always use the GetPoint, AddPoint, and ChangePoint routines. This way you will never have compatability problems.