PROGRAMMER'S CHALLENGE

By Bob Boonstra, Westford, MA

PROJECTION

For the Challenge this month, we return to the topic of computer graphics — you'll be solving a simplified rendering problem. Your Challenge is to create the image formed by a set of polygons on a specified projection plane, as viewed from a specific viewpoint, and as illuminated from a point light source. You will need to perform hidden surface elimination, create shadows caused by the light source, and project the image as it would be seen by someone at the viewpoint. You will be performing multiple projections from a given viewpoint, so this Challenge includes an initialization routine as well as a calculation routine, both of which are included for timing purposes in determining the winner.

The prototype for the code you should write is:

```
#define kMAXPOINTS 10

typedef struct My2DPoint {  /* point in z==0 plane */
  float x2D;                /* x coordinate */
  float y2D;                /* y coordinate */
} My2DPoint;

typedef struct My3DPoint {
  float x3D;               /* x coordinate */
  float y3D;               /* y coordinate */
  float z3D;               /* z coordinate */
} My3DPoint;

typedef struct My3DDirection {
  float thetaX;            /* angle in radians */
  float thetaY;            /* angle in radians */
  float thetaZ;            /* angle in radians */
} My3DDirection;

typedef struct MyPlane {
  My3DDirection planeNormal;   /* normal vector to plane */
  My3DPoint     planeOrigin;   /* origin of plane in 3D space */
} MyPlane;

typedef struct MyPolygon {
  long      numPoints;           /* number of points in polygon */
  My2DPoint thePoint[kMAXPOINTS];     /* polygon in z==0 plane */
  MyPlane   polyPlane;           /* rotate/translate z==0 plane to this plane */
  RGBColor  polyColor              /* the color to draw this polygon */
} MyPolygon;

void InitProjection(
  My3DPoint   *viewPoint,    /* viewpoint from which to project */
  My3DPoint   *illumPoint,     /* viewpoint from which to draw shadow */
  void        *storage,      /* auxiliary storage preallocated for your use */
  long       storageSize   /* number of bytes of storage */
);

void CalcProjection(
```

```
  GWorldPtr    offScreen,      /* GWorld to draw projection */
  MyPolygon    thePolys[],     /* polygons to project */
  long         numPolys,       /* number of polygons to project */
  My3DPoint    *viewPoint,     /* viewpoint from which to project */
  My3DPoint    *illumPoint,    /* illumination point from which to draw shadow */
  void         *storage,       /* auxiliary storage preallocated for your use */
  long         storageSize     /* number of bytes of storage */
);
```

Your InitProjection routine will be provided with a pointer to auxiliary storage (storageSize bytes, at least 1MB) preallocated for your use, along with the viewPoint from which projections are to be made and the illumPoint location of an illumination source from which shadows are to be created. InitProjection may perform any calculations that may be useful for multiple CalcProjection calls that follow. CalcProjection will be provided the same parameters given to InitProjection, along with the number (numPolys) and location of the polygons to be projected, and the offScreen GWorld in which the projection is to be drawn. CalcProjection should calculate the way thePolys would look from viewPoint, projected onto a projection plane normal to the viewPoint vector and passing through the origin. Hidden surface elimination must be performed so that obscured polygons or parts of polygons are not seen. The image of the projection is to be rendered in the GWorld pointed to by offScreen, with the projection plane mapped to the z==0 plane in the GWorld. Polygons must be rendered in the appropriate polyColor, subject to the limitations of the GWorld. Polygons are the same color on both sides. Parts of the projection plane not filled by projections of polygons should be black.

In addition to projecting the polygon image as seen from viewPoint, you must also project the shadow of thePolys created by an illumination source at illumPoint, onto the projection plane and onto the image of other polygons, as seen from viewPoint. Shadows should be rendered in the color of the surface in shadow, using a 50% gray pattern. All polygons have a flat matte surface, creating no specular reflections of the illumination source. The illumPoint will be on the same side of the projection plane as the viewPoint.

Polygons are specified in 2-dimensional coordinates in the z==0 plane, to ensure that all points are coplanar, along with a planeNormal vector that specifies the orientation of the polygon plane and a planeOrigin that specifies the plane origin. The last vertex of a polygon is connected to the first vertex to close the polygon (i.e., a square would have four vertices, not a fifth that is the same as the first.) The true polygon coordinates to be projected are calculated by first rotating counterclockwise about the positive z axis by thetaZ (i.e., the positive x axis rotated 90 degrees maps to the positive y axis), then counterclockwise about the positive x axis by thetaX (i.e., positive y rotates to positive z), then counterclockwise about the positive y axis by thetaY (i.e., positive z rotates to positive x), and finally by translating the origin to the planeOrigin point. In matrix form, the transformation is:

```
  | X |    | x3D |              | x2D |
  | Y | =  | y3D | + Ry Rx Rz   | y2D |, where
  | Z |    | z3D |              |  0  |

        | cos(thetaZ) -sin(thetaZ)        0      |
   Rz = | sin(thetaZ)  cos(thetaZ)        0      |
        |      0            0             1      |

        |      1            0             0      |
   Rx = |      0       cos(thetaX)  -sin(thetaX) |
        |      0       sin(thetaX)   cos(thetaX) |

        | cos(thetaY)       0         sin(thetaY) |
   Ry = |      0            1             0       |
```

```
|-sin(thetaY)        0       cos(thetaY) |
```

The offScreen GWorld will have a pixelDepth of 32. The viewPoint and illumPoint will have z coordinates greater than zero, but thePolys may have coordinates with arbitrary values (after rotating and translating the polyPlane). The projection plane is opaque, meaning that any part of a polygon behind the projection plane is invisible, creating no projection and no shadow.

On average, CalcProjection will be called approximately 10 times with the same viewpoint and illumPoint, but different polygons, for each call to InitProjection. The code producing the fastest projection, including both the InitProjection and CalcProjection times, will be the winner.

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

THREE MONTHS AGO WINNER

Perhaps it was the short amount of time to work with the BeOS CD-ROM bundled in the January issue of the magazine, or the fact that the BeOS required a 604 PowerMac, or some minor installation anomalies with the BeOS, or to migration of interest to a prospective Next-OS — whatever the reason, only two people entered the BeSort Challenge. Congratulations to **Charles Higgins** for submitting the fastest solution to the BeSort Challenge. The problem itself was fairly simple: write a SortWindow class that would sort a list of character strings by one of three methods, two specified by the problem statement and one of your own choosing.

Both Charles and the second contestant, **Kenneth Slezak**, implemented the required bubble sort and exchange sort methods, and both used the quicksort algorithm for the third method. The main difference in efficiency was in the technique used to swap list elements. Charles exchanged the pointers in the list and invalidated the list view to cause the list to be redrawn. Kenneth deleted the items to be exchanged from the list and added the items back into the list in the reverse order. On my 8500, the former was faster by 10+%. Interestingly enough, the latter was ~5% faster. Since the problem statement called for evaluation on the Macintosh, Charles' solution is the winner.

One other interesting observation — in the winning solution, execution time was dominated by display time. I verified this by repeating the timing tests with the windows hidden. In the winning solution, this reduced execution time by almost 80%. In Ken Slezak's solution, execution time was dominated by the list additions/deletions used to swap list elements, so the difference in results is much smaller.

A straightforward optimization to the winning solution improved execution time significantly. Instead of invalidating the ListView each time two elements were exchanged, one need only invalidate the rectangles for the two items being exchanged. This change reduced execution time by some 30% when the windows were visible. (It actually hurt performance when the windows were not visible.)

The table below provides the execution times and code sizes for each two solutions submitted, plus the optimized version of the winning solution. It shows the time, in seconds, required to sort a list of 500 strings by each of the three sort methods, with either visible windows or invisible windows.

	Visible Window				Invisible Window				
	TOTAL	Bubble	Xchg	Quick	TOTAL	Bubble	Xchg	Quick	Code
Charles Higgins	56.6	1.0	54.6	0.9	12.0	0.4	11.3	0.4	1472
Ken Slezak	64.0	0.8	59.7	3.4	59.8	0.7	55.9	3.2	1620
Optimized	38.5	0.7	37.0	0.8	33.9	0.5	32.7	0.7	1536

TOP 20 CONTESTANTS

Here are the Top Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	182	11.	Nicolle, Ludovic	21
2.	Gregg, Xan	114	12.	Picao, Miguel Cruz	21
3.	Larsson, Gustav	67	13.	Brown, Jorg	20
4.	Lengyel, Eric	40	14.	Gundrum, Eric	20
5.	Lewis, Peter	32	15.	Higgins, Charles	20
6.	Boring, Randy	27	16.	Kasparian, Raffi	20
7.	Cooper, Greg	27	17.	Slezak, Ken	20
8.	Antoniewicz, Andy	24	18.	Studer, Thomas	20
9.	Beith, Gary	24	19.	Karsh, Bill	19
10.	Cutts, Kevin	21	20.	Nevard, John	17

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points	5th place	2 points
2nd place	10 points	finding bug	2 points
3rd place	7 points	suggesting Challenge	2 points
4th place	4 points		

Here is Charles Higgins' winning solution:

SORTWINDOW.CPP
Charles Higgins

```
#include "SortWindow.h"

void swap(BWindow *aWindow, char **s1, char **s2);
char **addlist( BWindow *aWindow, char **list, int numberOfThings);

SortWindow::SortWindow(BRect frame)
         : BWindow(frame, "Sort", B_TITLED_WINDOW, 0)
{
   BRect            aRect = frame;
   BListView       *aView;

   aRect.OffsetTo(B_ORIGIN);
   aView = new BListView(aRect, "SortView",
                          B_FOLLOW_ALL, B_WILL_DRAW);
   this->AddChild(aView);
}

void swap(BWindow *aWindow, char **s1, char **s2)
{
   BView   *aView;
```

```
   char    *temp;

   aView = aWindow->FindView("SortView");
   aWindow->Lock();
   temp = *s1;
   *s1 = *s2;
   *s2 = temp;
   aView->Invalidate();
   aWindow->Unlock();
}

char **addlist( BWindow *aWindow, char **list, int numberOfThings)
{
   BListView       *aView;
   int              i;

   aView = (BListView*)aWindow->FindView("SortView");
   aWindow->Lock();
   for(i=0;i< numberOfThings;i++)
      aView->AddItem(list[i]);
   aWindow->Unlock();
   return((char**)aView->Items());
}

void SortWindow::DoSort(
   char *thingsToSort[], int numberOfThings, SortType sortMethod)
{
   short            i,
                    j,
                    k,
                    sorted = FALSE;
   char           **myList;

   myList = addlist( this, thingsToSort, numberOfThings);
   switch(sortMethod)
   {
      case kBubbleSort:
         i = numberOfThings-1;
         while(i>0)
         {
            j=i;
            for(k=0;k<i;++k)
            {
               if (0 < strcmp(myList[k],myList[j]))
                  j = k;
            }
            swap( this, &myList[i], &myList[j]);
            i--;
         }
         break;
      case kExchange:
         while(!sorted)
         {
            sorted = TRUE;
            for(i=0;i<numberOfThings-1;i++)
            {
               if(0 < strcmp(myList[i],myList[i+1]))
```

```
                {
                   sorted = FALSE;
                   swap( this, &myList[i], &myList[i+1]);
                }
             }
          }
          break;
       case kMySort:
          QuickSort( myList, 0, numberOfThings);
          break;
    }
    memcpy(thingsToSort,myList,numberOfThings*sizeof(char*));
    be_app->PostMessage(B_QUIT_REQUESTED);
}

void SortWindow::QuickSort( char **list, int first, int last)
{
    int              j,i;

    while(last - first > 1)
    {
       i = first;
       j = last;
       for(;;)
       {
          while(++i < last && strcmp(list[i],list[first]) < 0)
             ;
          while(--j > first && strcmp(list[j],list[first]) > 0)
             ;
          if (i >= j)
             break;
          swap( this, &list[i], &list[j]);
       }
       if( j == first)
       {
          ++first;
          continue;
       }
       swap( this, &list[first], &list[j]);
       if(j - first < last - (j+1))
       {
          QuickSort( list,first,j);
          first = j + 1;
       }
       else
       {
          QuickSort( list,j+1,last);
          last = j;
       }
    }
}
```

SortWindow.h

```
typedef enum SortType {
 kBubbleSort = 1,
```

```
 kExchange = 2,
 kExchangeSort = 2,
 kMySort = 3
 } SortType;

class SortWindow : public BWindow {

public:
      SortWindow(BRect frame);

virtual void DoSort( char *thingsToSort[],
     int numberOfThings,
     SortType sortMethod);

virtual void QuickSort( char **list, int first, int last);

};
```