Compression Study on *Pencil Test* Frames

A Kossow
Nov, 1989

Here are some numbers from running 640 X 480 8 bit RGB frames of Pencil Test through unix *compress*, encoding them in the proposed run format of Scooby, varying the number of bits saved (5 vs 6) and playing with the rendering quality. The size of an 8 bit frame was calculated by dividing the 555 run length format by two. The size of a 555 run length encoded frame was calculated by dividing by 3/4.

Files were compressed with *compress* to get a rough idea of how many patterns could be found in the input frame.

A second study investigated how much of a synthetic image could be represented by piece-wise linear interpolated runs (lrps). A lrp was defined as any horizontal run of pixels where the intensity continuously increases or decreases. Spans are conventional run-lengths, adjacent pixels on a scan line with the same color. An additional study needs to be done with some scanned images and non-teapot rendered synthetic images.

scene21-25.001

888 RGB

921600 640 * 480 * 3

22237	flatNoAA	41x
58905	flat33AA	16x
96405	smoothNoAA	10x
167439	smooth33AA	6.0x

664 RGB

614400 640 * 480 * 2

16238	flatNoAA	37x
29797	flat33AA	21x
33317	smoothNoAA	18x
55507	smooth33AA	11x

555 RGB

614400 640 * 480 * 2

16472	flatNoAA	37x
28892	flat33AA	21x
29591	smoothNoAA	21x
46644	smooth33AA	13x

Data Compression Using Unix 'Compress'

11 frames compressed individually and as a group

55805	scene21-25.100.IM24.Z
56214	scene21-25.101.IM24.Z
55968	scene21-25.102.IM24.Z
56180	scene21-25.103.IM24.Z
56585	scene21-25.104.IM24.Z
56636	scene21-25.105.IM24.Z
56670	scene21-25.106.IM24.Z
56805	scene21-25.107.IM24.Z
56741	scene21-25.108.IM24.Z
57060	scene21-25.109.IM24.Z
56675	scene21-25.110.IM24.Z

621339	16x	**total**
520291	19x	**compressed together**
10137600		**uncompressed**

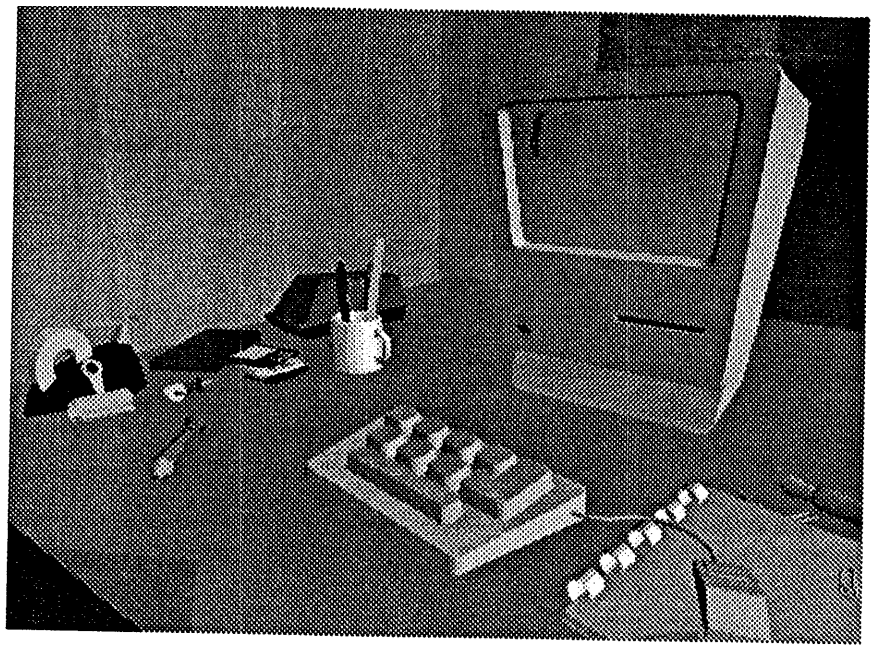
Comparing the Same Scene Rendered Differently

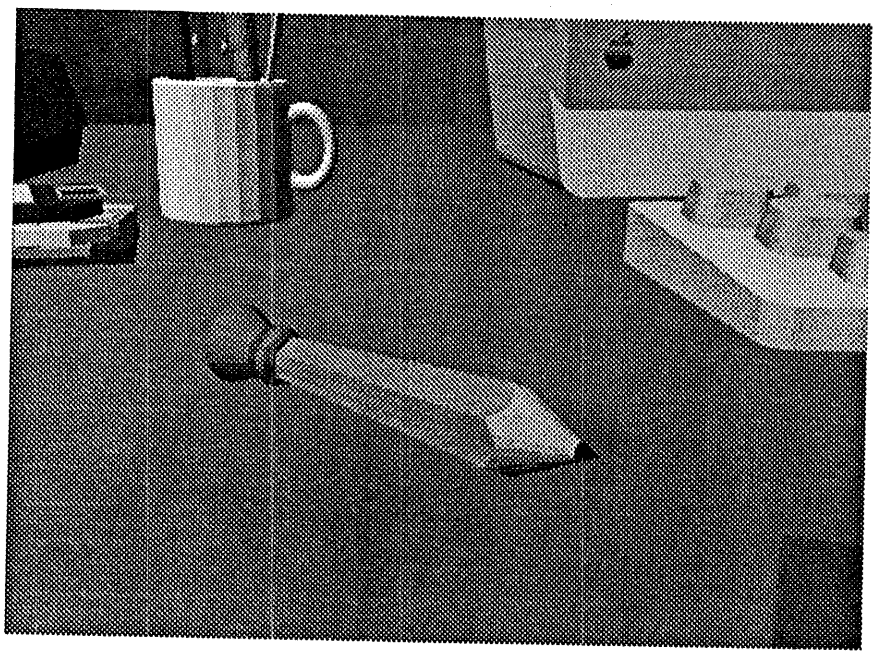
scene21-25.001	unix compress	555rle	555rle w/o alpha
flatNoAA	22237	30112	22584
flat33AA	58905	75712	56784
smoothNoAA	96405	89664	67248
smooth33AA	167439	143760	107820

Comparing Scenes in 'Pencil Test' Flat Shaded, AA

scene	unix compress	555rle		555rle w/o alpha		8bit w/o alpha	
05.001	106365	118640	7.8x	88980		59320	16x
06.001	84605	81248	11x	60936		40624	23x
07.001	46185	51488	18x	38616		25744	36x
08.001	29364	33232	28x	24924		16616	55x
10.001	25070	40032	23x	30024		20016	46x
18.001	22095	30960	30x	23220		15480	60x
19.001	79739	68704	13x	51528		34352	27x
20.001	22606	32256	29x	24192		16128	57x
21.001	58905	75712	12x	56784		37856	24x

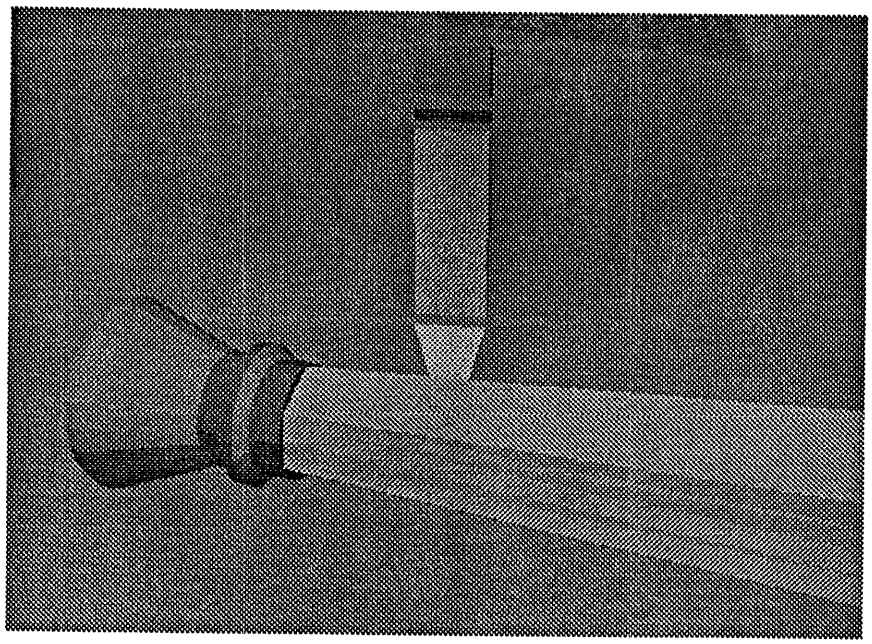
Scene 5



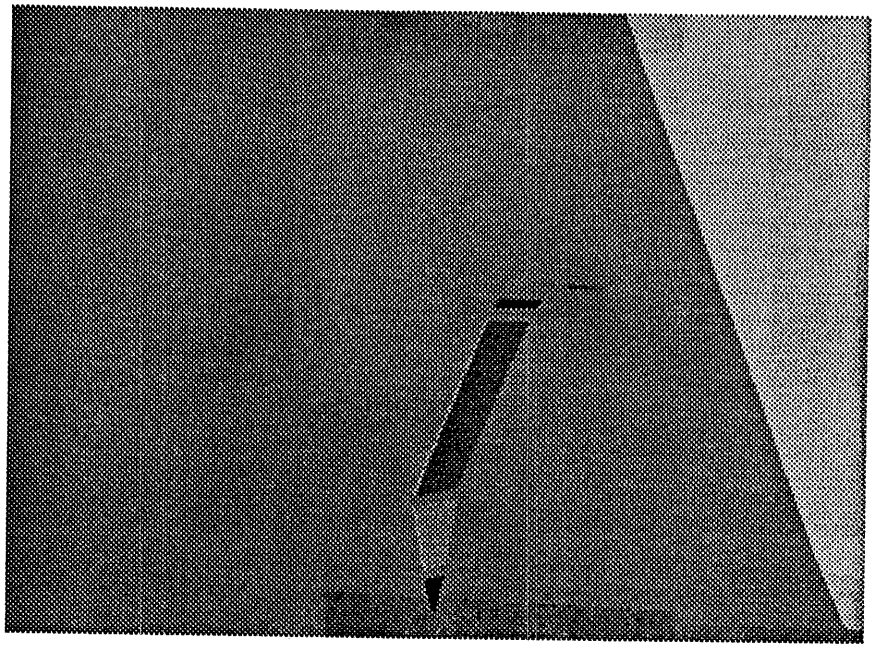


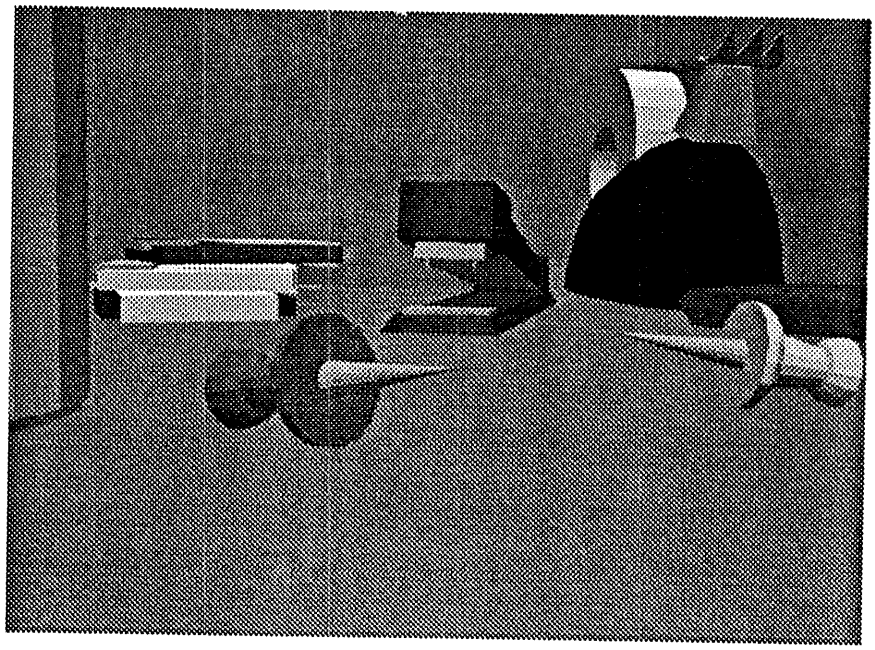
Scene 6

Scene 7



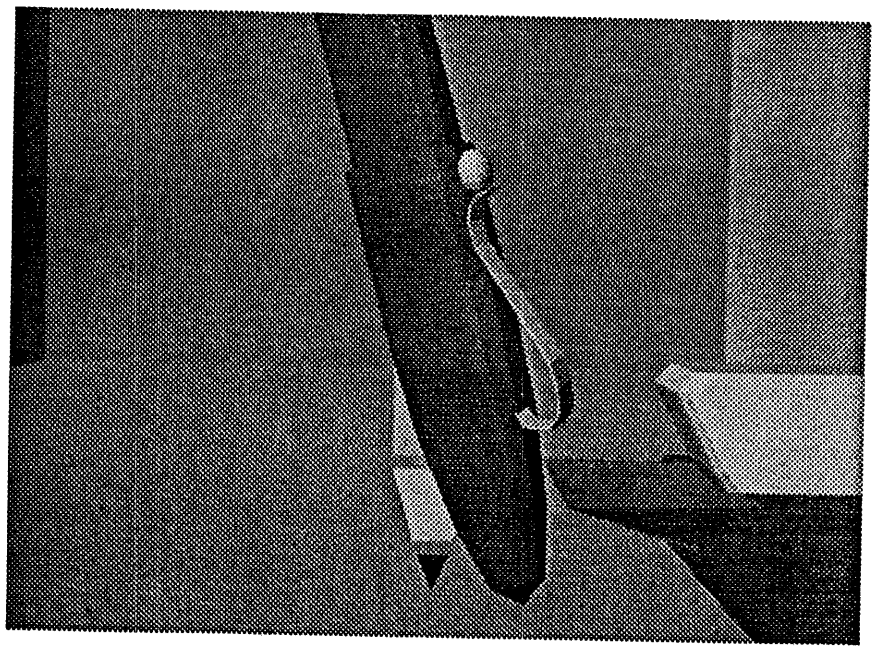
Scene 18





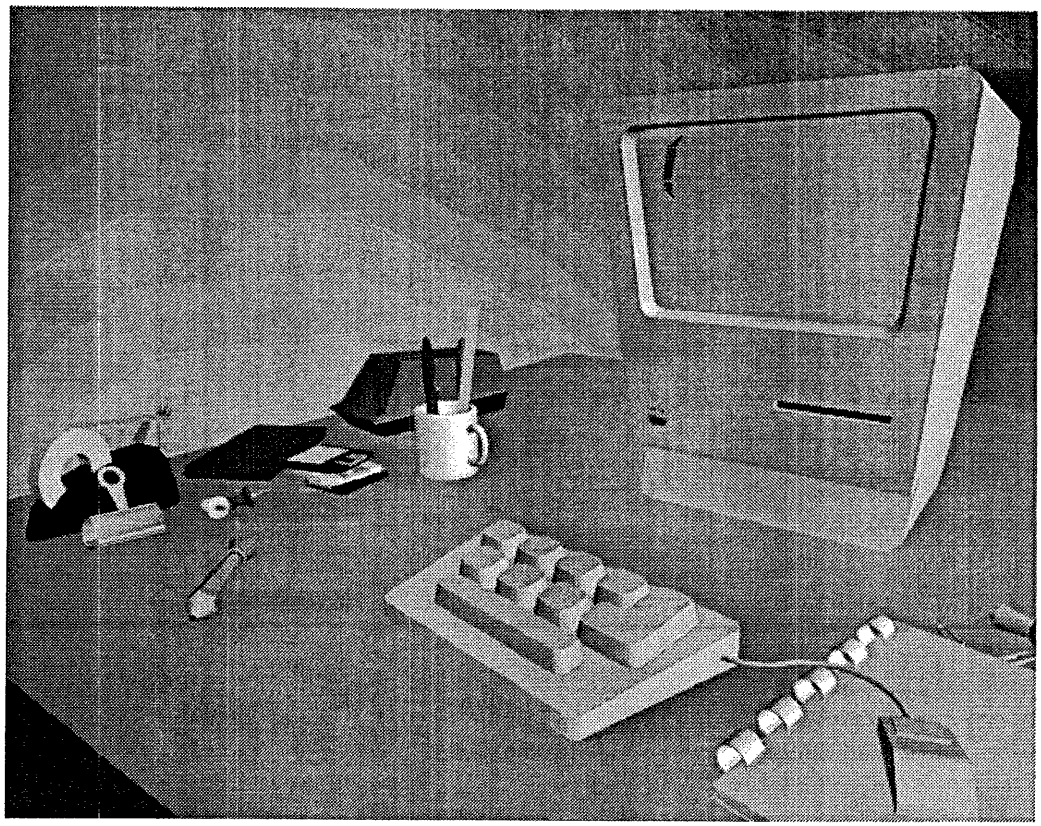
Scene 19

Scene 21

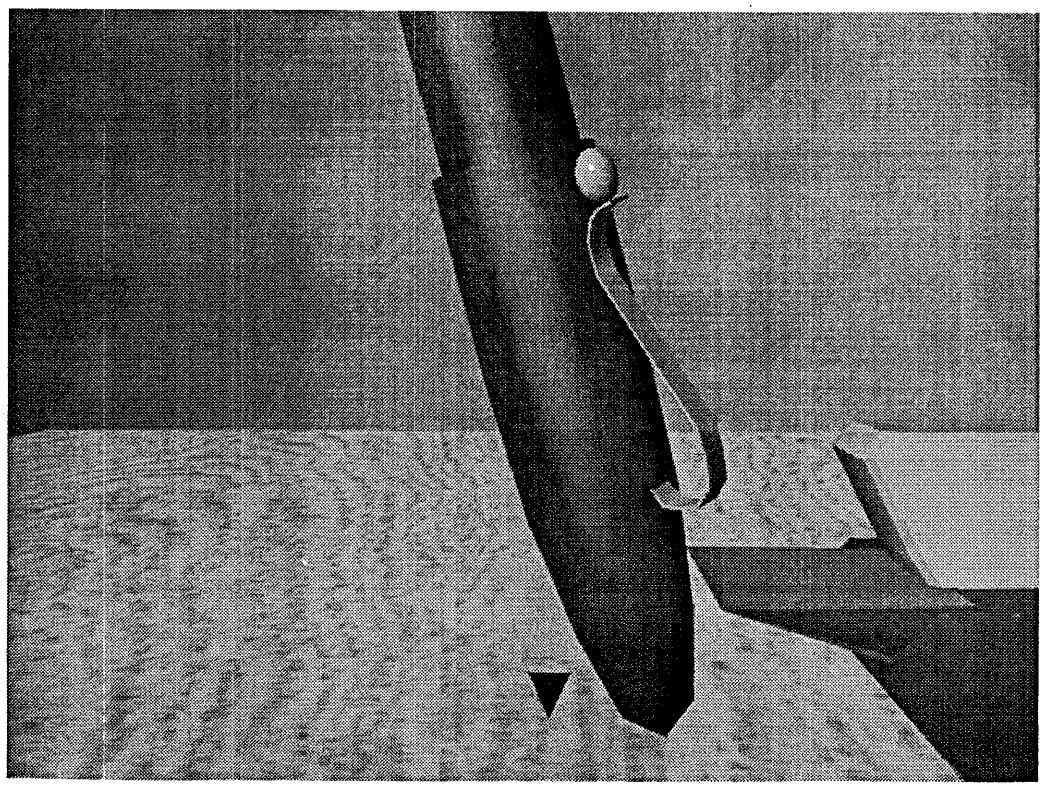


Number Per Frame	2	5	10	25	50	75	100	200	>200
scene21.phongTmAA(555)									
spans	48415	21798	11167	6201	1515	968	884	1982	87
lrps	99148	4840	597	11					
scene21.phongTmAA(666)									
spans	62774	19627	6934	6766	4280	2597	291	310	
lrps	102549	18364	3172	821	114				
scene21.smoothAA (555)									
spans	12429	3274	3283	4972	1895	1448	989	2146	711
lrps	35690	1541	307	9					
scene21.smoothAA (666)									
spans	15152	5764	5327	7343	4742	2742	396	455	623
lrps	52283	2078	533	577	104				
scene21.flatAA (555)									
spans	3143	1133	2693	3879	1054	550	375	2030	1631
lrps	22117	1367	43						
scene21.flatAA (666)									
spans	3558	1301	3227	3934	1294	240	526	1853	1623
lrps	28013	2951	131	2					
scene21.flatNoAA (666)									
spans	3009	1116	3188	4053	1221	419	374	2019	1668
lrps	16343	18							
scene21.flatNoAA (555)									
spans	2958	1076	3033	3810	1034	622	374	2019	1668
lrps	15862	18							
scene05.smoothAA (555)									
spans	11717	4504	2661	4592	2783	1745	889	1547	769
lrps	37057	5963	790	91					
scene05.smoothAA (666)									
spans	16859	5892	3524	7779	3365	1031	831	1550	493
lrps	53111	7103	1325	304	13				
scene05.flatAA (555)									
spans	7904	3134	1789	3935	1948	1882	876	2575	522
lrps	31351	6054	321						
scene05.flatAA (666)									
spans	9164	3161	1782	3954	1797	1885	851	2587	481
lrps	37844	8213	756	19					

Total Number of Runs and Lrps Per Frame
Independent RGB Channels



scene05. smooth AA



scene21phong TmAA

```
#include <stdio.h>

unsigned char linebuf[640][3];

int spanCnt = 0,lrpCnt = 0;
int spanBkt[10];
int lrpBkt[10];

#define SPAN 0
#define LRP  1

#define RED  0
#define GRN  1
#define BLU  2

main()
{
        int line=1,total=0;
        register int i;
        unsigned char iniR, iniG, iniB;
        int accumR, accumG, accumB;
        int runcntR, runcntG, runcntB;
        int dR,dG,dB;

        while(fread(linebuf,3,640,stdin) != 0){

        /*
         * round up all pixels to 5 bits
         */
         for(i=0; i<640; i++){
          linebuf[i][0] = ((linebuf[i][0] -1) + 7) & 0xf8;
          linebuf[i][1] = ((linebuf[i][1] -1) + 7) & 0xf8;
          linebuf[i][2] = ((linebuf[i][2] -1) + 7) & 0xf8;
         }

/*       printf("\nLINE %d\n",line++); */

         runcntR = runcntG = runcntB = 0;
         accumR  = accumG  = accumB  = 0;
         iniR = iniG = iniB = 0;

          dR = (linebuf[1][0] ) - (linebuf[0][0] );
          dG = (linebuf[1][1] ) - (linebuf[0][1] );
          dB = (linebuf[1][2] ) - (linebuf[0][2] );

         if(dR == 0)
          runcntR = 1;
         else
          accumR = dR;

         if(dG == 0)
          runcntG = 1;
         else
          accumG = dG;

         if(dB == 0)
          runcntB = 1;
         else
          accumB = dB;
/*
 *
 *
 */
         for(i=1; i<640; i++){
```

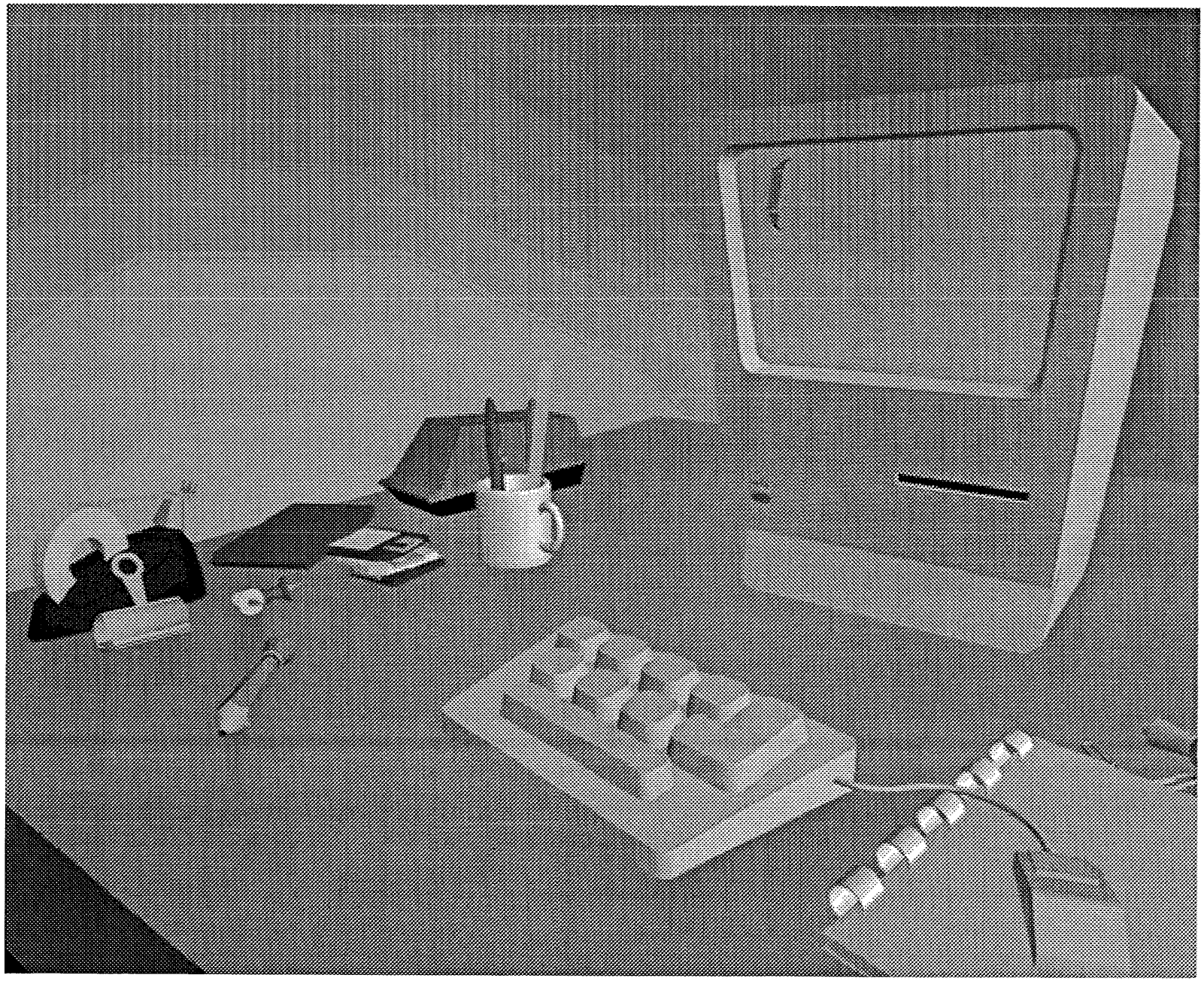
```
          dR = (linebuf[i+1][0] ) - (linebuf[i][0] );
          dG = (linebuf[i+1][1] ) - (linebuf[i][1] );
          dB = (linebuf[i+1][2] ) - (linebuf[i][2] );

/*printf("pixel %d RGB %x %x %x Dclr %d %d %d\n",
        i,linebuf[i][0],linebuf[i][1],linebuf[i][2],dR,dG,dB); */
/*
 *
 *  *   *
 *   \ /
 *  *-*-*
 *   / \
 *  *   *
 *
 */
          /*                    *
           *                   /
           * handle runs    *-*-*
           *                  \
           *                   *
           */
          if(accumR == 0){                          /*       *      */
           if(dR != 0){                             /*      /       */
            doOutput(RED,SPAN,iniR,runcntR);        /*  *-*         */
            iniR = i;                               /*      \       */
            runcntR = 0;                            /*       *      */
            accumR = dR;
           }
           else{
                runcntR++;                          /*  *-*-*       */
           }
          }

          /*              *   *
           *               \ /
           * handle lrps    *-*
           *                / \
           *              *   *
           */
          else{
           if(dR == 0){                                 /*     *         */
                doOutput(RED,LRP,accumR,i-iniR);        /*      \        */
                accumR = 0;                             /*       *-*     */
                iniR = i;                               /*      /        */
                runcntR = 1;                            /*     *         */
           }
           else{

                if(dR > 0){                             /*     *   *     */
                 if(accumR > 0){                        /*      \ /      */
                  accumR += dR;                         /*       *       */
                 }                                      /*      /        */
                                                        /*     *         */
                 else{
                  doOutput(RED,LRP,accumR,i-iniR);
                  iniR = i;
                  accumR = dR;
                 }
                }

                else{                                   /*      *         */
                 if(accumR < 0){                        /*       \        */
                  accumR += dR;                         /*        *       */
                 }                                      /*       / \      */
                                                        /*      *   *     */
```

```
                else{
                 doOutput(RED,LRP,accumR,i-iniR);
                 iniR = i;
                 accumR = dR;
                }
               }
              }
             }
/*
 *
 */
            if(accumG == 0){
             if(dG != 0){
                  doOutput(GRN,SPAN,iniG,runcntG);
                  iniG = i;
                  runcntG = 0;
              accumG = dG;
             }
             else{
                  runcntG++;
             }
            }
            else{
             if(dG == 0){
                  doOutput(GRN,LRP,accumG,i-iniG);
                  accumG = 0;
                  iniG = i;
                  runcntG = 1;
             }
             else{
                  if(dG > 0){
                   if(accumG > 0){
                    accumG += dG;
                   }
                   else{
                    doOutput(GRN,LRP,accumG,i-iniG);
                    iniG = i;
                    accumG = dG;
                   }
                  }
                  else{
                   if(accumG < 0){
                    accumG += dG;
                   }
                   else{
                    doOutput(GRN,LRP,accumG,i-iniG);
                    iniG = i;
                    accumG = dG;
                   }
                  }
             }
            }
/*
 *
 */
            if(accumB == 0){
             if(dB != 0){
                  doOutput(BLU,SPAN,iniG,runcntB);
                  iniB = i;
                  runcntB = 0;
              accumB = dB;
             }
             else{
                  runcntB++;
             }
```

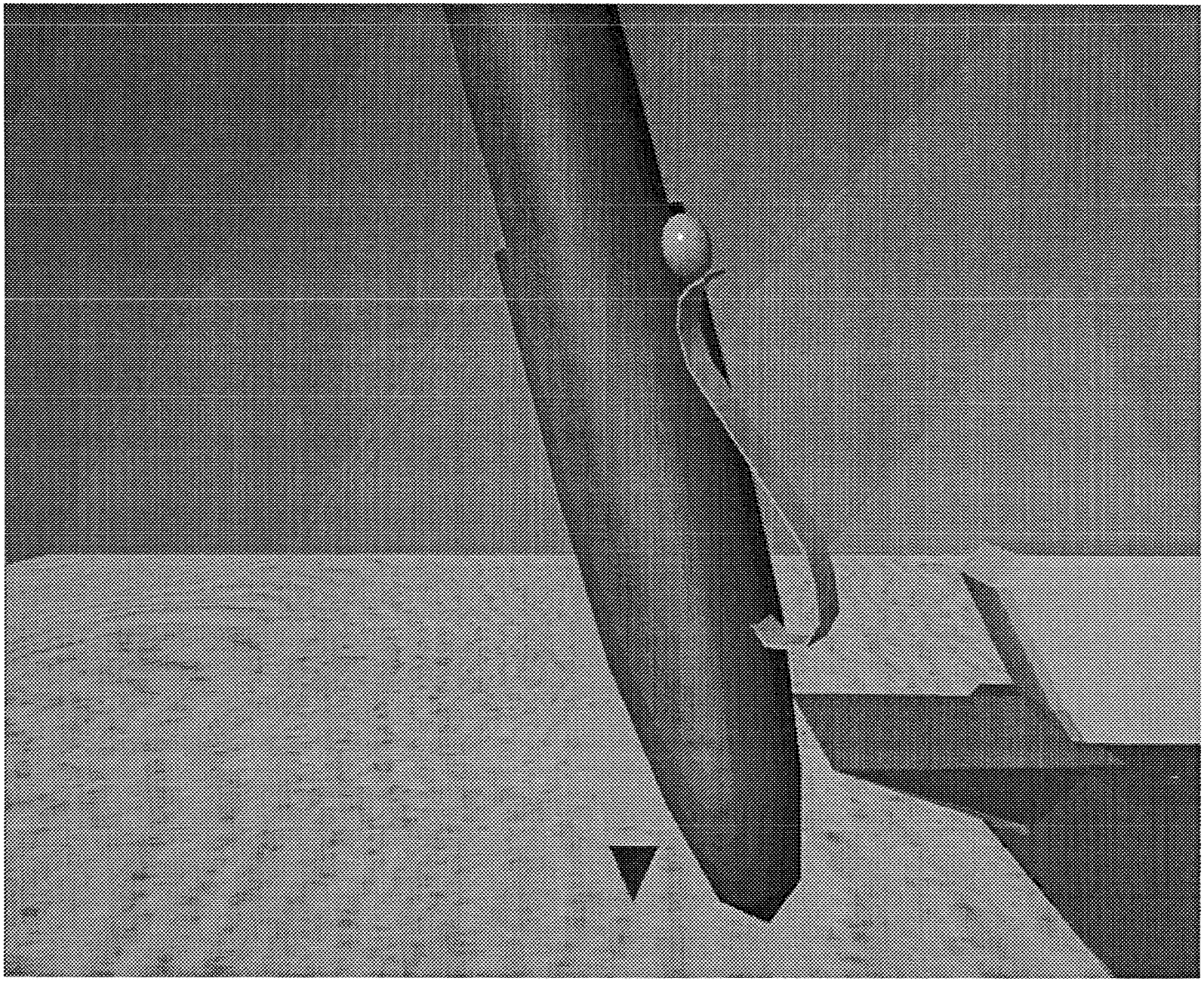
```
            }
            else{
             if(dB == 0){
                  doOutput(BLU,LRP,accumB,i-iniB);
                  accumB = 0;
                  iniB = i;
                  runcntB = 1;
             }
             else{
                  if(dB > 0){
                   if(accumB > 0){
                    accumB += dB;
                   }
                   else{
                    doOutput(BLU,LRP,accumB,i-iniB);
                    iniB = i;
                    accumB = dB;
                   }
                  }
                  else{
                   if(accumB < 0){
                    accumB += dB;
                   }
                   else{
                    doOutput(BLU,LRP,accumB,i-iniB);
                    iniB = i;
                    accumB = dB;
                   }
                  }
             }
            }
           }
          }
         printf("%d spans %d lrps\n",spanCnt,lrpCnt);

         printf("span bkts %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d\n",
          spanBkt[0],spanBkt[1],spanBkt[2],spanBkt[3],spanBkt[4],
          spanBkt[5],spanBkt[6],spanBkt[7],spanBkt[8],spanBkt[9] );

         printf("lrp bkts  %4d %4d %4d %4d %4d %4d %4d %4d %4d %4d\n",
          lrpBkt[0],lrpBkt[1],lrpBkt[2],lrpBkt[3],lrpBkt[4],
          lrpBkt[5],lrpBkt[6],lrpBkt[7],lrpBkt[8],lrpBkt[9] );
}


doOutput(color,type,accum,len)
{
/*
         switch(color){
          case(RED):
           printf("RED: ");
           break;
          case(GRN):
           printf("GRN: ");
           break;
          case(BLU):
           printf("BLU: ");
           break;
         }
*/
         switch(type){
          case(SPAN):
           spanCnt++;

           if(len < 10)
```

```
            spanBkt[0]++;
           else
            if(len < 20)
             spanBkt[1]++;
            else
             if(len < 30)
              spanBkt[2]++;
             else
              if(len <40)
               spanBkt[3]++;
              else
               if(len <50)
                spanBkt[4]++;
               else
                if(len < 60)
                 spanBkt[5]++;
                else
                 if(len < 70)
                  spanBkt[6]++;
                 else
                  if(len < 80)
                   spanBkt[7]++;
                  else
                   if(len < 90)
                    spanBkt[8]++;
                   else{
                    printf("spanlen = %d\n",len);
                    spanBkt[9]++;
                   }

/*          printf(" span %d len %d\n",accum,len); */
           break;
           case(LRP):
            lrpCnt++;

            if(len < 10)
             lrpBkt[0]++;
            else
             if(len < 20)
              lrpBkt[1]++;
             else
              if(len < 30)
               lrpBkt[2]++;
              else
               if(len <40)
                lrpBkt[3]++;
               else
                if(len <50)
                 lrpBkt[4]++;
                else
                 if(len <60)
                  lrpBkt[5]++;
                 else
                  if(len < 70)
                   lrpBkt[6]++;
                  else
                   if(len < 80)
                    lrpBkt[7]++;
                   else
                    if(len < 90)
                     lrpBkt[8]++;
                    else{
                     printf("lrplen = %d\n",len);
                     lrpBkt[9]++;
                    }
```

```
/*          printf(" lrp  %d len %d\n",accum,len); */
           break;
          }
}
```



scene 05. smooth AA



Scene21phongTmAA