The Demoscene

It is said that mathematics, which includes computer science, is an area involving creativity, like art, or a similar, intuitive, process oriented towards discovery but that is much like art. I would like to introduce and describe a computer subculture in which arts and hacking are combined.

The *demoscene* (scene) is a computer art subculture whose members create demonstrations (demos). Thomas Gruetzmacher's PC Demoscene FAQ states[1]:

> "2.2. What is the demoscene?
>
> Short answer: A subculture in the computer underground culture universe, dealing with the creative and constructive side of technology, proving that a computer can be used for much more than writing a letter in MS-Word and hence emphasize [sic] on computer technology as just another medium that can transport ideas and styles, show off skills and express opinions etc.
>
> Another theory says, that it's just a bunch of boozing computer nerds, programming weird, useless multimedia stuff. Errm. ;)"

The alt.sys.amiga.demos Usenet newsgroup FAQ states[2]:

> "Demos, (short for 'demonstrations'), are executable programs created (in the case of this FAQ, on the Amiga computer), purely for art's sake, featuring impressive or spectacular audiovisuals. Demos are not actually functional or interactive, in the main, but then nor are portraits, or CDs. Perhaps you can think of a demo as a music video on a computer, but with equal emphasis on the visuals, the music, and the code. It's something to watch, enjoy, and marvel at the creativity of. Demos can be beautiful."

The comp.sys.ibm.pc.demos newsgroup FAQ states[3]:

> "A Demo is a program that displays a sound, music, and light show, usually in 3D. Demos are very fun to watch, because they seemingly do things that aren't possible on the machine they were programmed on.
>
> Essentially, demos "show off". They do so in usually one, two, or all three of three following methods:
>
> They show off the computer's hardware abilities (3D objects, multi-channel sound, etc.)
> They show off the creative abilities of the demo group (artists, musicians)
> They show off the programmer's abilities (fast 3D shaded polygons, complex motion, etc.)

> Demos are an art form. They blend mathematics, programming skill, and creativity into something incredible to watch and listen to."

Thomas Gruetzmacher's PC Demoscene FAQ states[1], "Ultimately, a demo(nstration) in a scene sense, is a piece of free software that shows realtime rendered graphics while playing music. Often the music is tightly connected/synced to the visuals". A member of the demoscene is a *demoscener* (scener*)*.

Demos are similar to the *display hacks* (graphics demos) that started in the 1950s such as the bouncing ball one on the Whirlwind computer[4]. However, the scene started by youths and interested people, mostly in Europe, particularly the North, in the 1980s, has its origin in the software piracy/cracking subculture. On early eight-bit personal computers, such as the Commodore 64 (C64), Amstrad CPC, ZX Spectrum, Atari 800, and later the IBM-PC and compatibles (PC), crackers (in this sense, meaning people who break software copy protection—not necessarily other security), typically cracking games, would add introductions, or *crack intros (*cracktros*),* to the beginning of software. The cracktros initially listed the creators' names, perhaps showed a short message or few seconds of graphics/art and maybe audio/music, but soon intros grew longer. Eventually, intros started to be created for their own sake and then they, or larger productions, were called demos. Productions in between those sizes are *dentros*, and large demos are *megademos*. Demos with audio/music are *trackmos*. There are also art slideshows and *musicdisks* (scene albums which used to be released on floppy disk). In recent decades, cracktros have become rare, and most demos are made for art's sake. With the release of 16-bit computers such as the Commodore Amiga, Atari ST, and other IBM PCs, demos continued onto those, and 32-bit, 64-bit computers, as well as many game consoles, and Ti and probably HP and other graphing calculators.

There are several roles in the scene, and members usually form *demogroups*. Designers think about how a demo should look or sound—rather like a director. Coders/programmers program demos and other scene software. Artists/*graphicians*, such as traditional artists, ASCII/ANSI artists, *pixellers* (two-dimensional digital art makers) and three-dimensional (3D) graphics renderers, such as *tracers* (people who raytrace, or use software that accurately simulates light in space and on objects), animators, and even people who film, make demo art. Musicians/*trackers* use music module trackers (software to arrange music notes in a matrix of tracks and rows, or a MOD), or sometimes also software and hardware synthesizers, or the musicians' recorded instruments, to make demo music, and a large amount of unrelated electronic music are in newer MOD formats. Group members also write text to be displayed in demos. Disk magazine (*diskmag*) writers and web site masters write about the scene. System operators (*sysops)* used to, and a few still, operate bulletin-board systems (BBSes), but their role has mostly been replaced by volunteers/administrators of various Internet services, such as Usenet, Internet relay chat (IRC), both of which are still somewhat used in the scene, and the web (most sceners have migrated to non-Usenet forums, but a few still use IRC). Another role, probably obsolete, was that of *couriers*, who traded demos on BBSes and through mail on floppy disks. People also organize *demoparties*. Of course, sceners may have more than one role. They still use aliases/handles as a relic of the software piracy subculture, but also for online. Some "Renaissance man" sceners, such as Tran and Statix, took all the creative roles—producing their own trackmos single-handedly.

The first demogroups were The Judges, which started in 1986, and 1001 Crew—both from the Netherlands. The Judges released the C64 *Think Twice* demo series and *Rhaa Lovely* slideshows. The *Think Twice* series used the flexible line distance effect, which changes the distance between rows of text on the screen and makes them appear to "bounce". The first non-cracker demogroup was Razor 1911, which started in 1986, but in 1987 they started cracking games[5, pp 168]. C64 demo music is made for its SID chip that can do three types of sound waves, but since 1987, starting on the Commodore Amiga, most demo music is MOD or later, derived, formats (notably S3M, XM, and IT, the latter two of which are still widely used on the PC). The most famous demo is probably Future Crew's PC demo *Second Reality*, which was released at the Assembly 93 demoparty, and is long, with excellent design, with about 15 parts, including a secret one. Another of the most impressive demos, also released in 1993, at The Computer Crossroads demoparty, is Triton's Crystal Dream II, with about 13 parts. Both demos have 3D parts, and Crystal Dream is considered by many programmers to be more impressive because of its complex 3D scenes and zoom into the Mandelbrot set.

The height of the PC demoscene was probably 1995, when Complex's *Dope* was released at The Gathering demoparty. Dope's graphics were very advanced, and today's computer graphics do not seem much more impressive/realistic. Until about that time, many demos were more advanced than video games of the time. At about that time, demos began using effects of 3D graphics cards rather than just VGA cards, to the disappointment of some programmers who prefer doing all the details of graphics themselves. Also, many Amiga and PC intros were programmed in pure assembly language.

Demos are often released at demo competitions (*democompos*, *compos*) at demoparties. These happened in the years after the software piracy subculture's *copyparties* started. The largest demoparties have been held in Northern Europe, and include The Gathering, which ran from 1991 to 2001 each Spring in Norway, the Assembly, which has run since 1992 each August in Finland, and The Party, which has run since 1994 each Winter in Denmark. The Evoke demoparty has been held in Germany since 1997. There have been several demoparties in North America, such as the early North American International Demoparty (in Canada in the early to mid 1990s), Spring Break (in California in the mid to late 1990s), Pilgrimage (in Utah, which I attended in 2003), and a few newer ones. Demoparties have been held in many countries all over the world. A demo inviting people to a demoparty is an *invitro*. Demoparties also have computer art & music compos and other events. Demoparties grew over the years, and in the late 1990s, more video gamers attended, and so the role of networked computer games increased, almost taking over the purpose of some parties. As the scene grew, parties specifically for art or music started. Probably sometime after the Internet became public, online compos started, and some parties have allowed remote submission of entries.

Many demo effects exist. Any computer graphics or electronic audio technique can be a demo effect. An Intro might just have one, but demos usually have several visual scenes and one or more pieces of art or music. The mathematics of Euclidean, fractal, and possibly other geometry, trigonometry, analysis/calculus, and linear algebra are used to program demo effects. *Plasma* is an effect of which several types exist: colored-in cloud fractals, and trigonometric functions used to make wavy effects, or that in 3D look like smoke or steam. There are effects to make realistic-looking water and fire. A *shadebob* is

an effect in which a small shape, usually circular, moves through the screen and changes the color. Usually, multiple shadebobs are done, which appears similar to plasma. A *rasterbar* is an old effect displaying a horizontal bar of color, which relies on an electron beam in a CRT returning to the left to begin a new scanline. The same visual effect can be achieved on LCDs, but for LCDs, it is not quite a hack as it is with CRTs. Edwin Catmull's team at University of Utah discovered how to hide 3D surfaces that are behind other surfaces, which also enabled coloring surfacesp. Catmull's method is *z-buffering*[5, pp 25][6]. Z-buffering is used in many 3D demos. Foley, et al, in their book recommended by the comp.sys.ibm.pc.demos FAQ, give C-style pseudocode for z-buffering as follows[7]. It assumes one is using their C library with the functions WritePixel (like the perhaps more common put_pixel()), and WriteZ & ReadZ for writing & reading z-buffers.

```
“void zBuffer(void)
{
  int x,y;
  for(y=0;y<YMAX;y++){ /* Clear frame buffer and z-buffer
*/
    for(x=0;x<XMAX;x++){
      WritePixel(x,y,BACKGROUND_VALUE);
      WriteZ(x,y,0);
     }
   }
   for(each polygon){ /* Draw polygons */
     for(each pixel in polygon's projection){
       double pz=polygon's z-value at pixel coords(x,y);
       if(pz>ReadZ(x,y){ /* New point is not farther */
         WriteZ(x,y,pz);
         WritePixel(x,y,polygon's color at pixel coords
(x,y))
       }
     }
  }
} /* z-buffer */”
```

In 1971, Henri Gouraud discovered a 3D shading method that made curved surfaces appear more realistic. [5, pp 26][8]. Gouraud shading is used in many 3D demos. It interpolates a tone of shade/light intensity along each scanline in a polygon, creating a gradient (gradual shading and lighting) along each line. The algorithm, in functional pseudocode, is as follows.

```
Include a function, frac(), to return a number's
fractional part.
Define a polygon.
For i=top of polygon to number of scanlines:
  Define ax,bx,cx,dx as x-values at points a,b,c,d.
  Define atone,btone as tones at points a,b.
```

```
  Let gradient=(btone-atone)/(bx-ax).
  Let ctone=at+(1-frac(ax))*gradient.
  For j=cx to dx:
    Put pixel at (x,y) with colour ctone.
    ctone=ctone+gradient.
  End for.
End for.
```

To shade/light an actual 3D object, presumably there would be an additional first loop for i (not the same i as for the polygons) to the number of polygons, and the pixel values would be saved in a memory buffer for the screen, which would be output all at once afterwards, rather than putting pixels in the actual algorithm. Otherwise, if the pixels must be output in the algorithm, the one for 3D objects must be a bit more complex. In 1971, Bui-Tong Phong discovered a more advanced shading method, based on Gouraud shading and involving vectors[5][9].

In 1974, Catmull also discovered texture-mapping, and in 1976 Jim Blinn wrote an advanced article on it[5, pp 27][10]. Denthor of the Asphyxia demogroup gives the following Pascal texture-mapping code in one tutorial[11]. It is for a square in a diamond orientation.(i.e., with one vertex at the top) and a texture 128 pixels wide.

```
“textureX = 0;
textureY = 64;
textureEndX = 64;
textureEndY = 0;

dx := (TextureEndX-TextureX)/(maxx-minx);
dy := (TextureEndY-TextureY)/(maxx-minx);
for loop1 := minx to maxx do BEGIN
  PutPixel (loop1, ypos, texture [textureX, textureY],
VGA);
  textureX = textureX + dx;
  textureY = textureY + dy;
END;”
```

Jim Blinn discovered *bump-mapping*, which simulates bumps and pits on 3D surfaces[5, pp 27]. A display hack/intro by HELiX gives the following Pascal bump mapping code[12].

```
“procedure dobump; {Now u guess what this one
does..hehehe}
var difx,   {The X axis difference}
    dify,   {The Y axis difference}
    col:byte; {Used in many points..}
    l1:integer; {General use}
begin
  lx:=160; {The starting position of the light source}
  ly:=100; {>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>}
```

```
  l1:=0;
  repeat
    inc(l1);
    lx:=round(cos(l1/13)*66+160);
    ly:=round(sin(l1/23)*66+100);
    {^^^^ those two make sure the light moves in a nice
round
     path..}
    for x:=1 to 319 do
      for y:=1 to 190 do begin
      {This were the important stuff is done}
      {Here we will light the point x,y if lx,ly is the
light
       position}

        vlx:=x-lx; {Calculate the L vector}
        vly:=y-ly; {>>>>>>>>>>>>>>>>>>>>>}

        if (vlx<130) and (vlx>-130) and (vly>-130) and
(vly<130 ) then begin
        {This is some stupid way to gain speed.. if the
light
         vector is too far away from the point we want to
light
         then don't bother..it will probably get no
light..}
          nx:=mem[vaddr:x+y*320+1]-mem[vaddr:x+y*320-1];
          ny:=mem[vaddr:x+(y+1)*320]-mem[vaddr:x+(y-
1)*320];
{Those two lines are the heart of bumping}
{We have a pixel, say x,y and we want to find how its
normal
 vector is facing.. (that is its slope...) normaly we
would have
 to mess with cos and sin and other shitty stuff but here
we
 only care about something like a pseudo-normal.. in
other words
 we only care about the Sign.. that is there are three
possible
 pseudo normals:
      1) nx<0 (Normal facing right)
    nx>0 (Normal facing left)
    nx=0 (Normal sticking out of the screen)
 and its the same story with ny.. (it can be facing
up,down or
 sticking out..}
{To find this orientation we get 2 pixels.. the one left
```

```
and the
 one right (or the one up and the one down for ny) and we
sub..
 the result is the N vector for our point 10,10}

{The rest is easy.. we have two vectors now.. N,L.. we
want
 their coordinates to be as close as possible (the closer
they
 are the more light gets the pixel}

          col:=abs(vlx-nx);
          {that is what I just said written in
mathematics
           hehehe}
          if col>127 then col:=127;
          {Just not to overflow}
          difx:=127-col;
          {^^^^ that is the first component of the final
color..
           the light we get from the X axis}
          if difx<0 then difx:=1;

          {Now we do the same stuff for the Y axis }
          col:=abs(vly-ny);
         if col>127 then col:=127;
         dify:=127-col;
        if dify<0 then dify:=1;
        {finaly we add the two intensities and we're
done..}
        col:=(difx+dify) ;
        if col>128 then
          mem[vaddr2:x+y*320]:=col;
          {That's it.. put the damn pixel}
       {putpixel(x,y,col,vaddr2);}
     end;
    end;
   flip32(vaddr2,sega000);
   cls32(vaddr2,0);
  until keypressed;
end;"
```

There are many 3D demo effects that are more complicated. These include *environment mapping*, discovered by Blinn, in which an object reflects its environment[5, pp 27][13], and he discovered a more advanced shading method. Also, in 1977, Rob Cook discovered a more advanced shading method that takes external light into account, and there are many newer shading methods[5, pp 28]. In 1968, Arthur Appel discovered

raycasting, i.e. basic raytracing[7, pp 701][14], and in 1980, Turner Whitted discovered more advanced raytracing[5, pp 28][15]. In 1948, Parry Moon and Eberle Spencer, discovered and plotted radiosity (on paper), which simulates photons, and in 1984, Cindy Goral, at Cornell University implemented it in raytracing[5, pp 28][16]. Volumetric pixels, or *voxels*, are objects plotted by coloring in polygons. Other effects include *lens flares* (bright areas of light through glass), *starfields*, realistic and abstract tunnels (some, such as the *Syn2x* display hack, of which can cause optical effects similar to hallucinations lasting for many seconds), and *vector balls* (balls, like points, making vertices and shapes).

Many demo programmers have gone on to work in industry, so there are commercially-made demos. There are also demo-generator programs. Despite that these programs and 3D graphics cards, etc., make demo creation easier, that allows more focus on the design, so the scene's future should be interesting.

Sometimes hackers need to have fun, such as art. I enjoy the demoscene, and hope you do, whether you watch or have watched a demo, or if you program, draw, compose for demos or the related arts scenes. Happy Hacking!

Bibliography

[1] T. Gruetzmacher (2004, Jun. 12) PC Demoscene FAQ [Online]. Available: http://tomaes.32x.de/text/faq.php
[2] S. Carless. (1996, Jul. 17). alt.sys.amiga.demos FAQ (1.08) [Online]. Available Usenet: nntp://alt.amiga.demos
[3] J. Leonard. (1988, Mar. 12). PC Demos FAQ (2.02) [online]. Available: http://www.oldskool.org/demos/pc/pcdemos.faq
[4] Viznut. (2006, Jul. 26) *Display* hack. [Online]. Available: http://en.wikipedia.org/wiki/Display_hack
[5] T. Polgár. *Freax: The Brief History of the Computer Demoscene*. Germany: CSW-Verlag, 2008.
[6] E. Catmull. "A Subdivision Algorithm for Computer Display of Curved Surfaces,"Ph.D. dissertation, CS Dept., Univ. of Utah, Salt Lake City, Utah, 1968.
[7] J. Foley et al. “Visible Surface Determination,” in Computer Graphics: Principles And Practice, 2nd ed.Addison-Wesley, 1997., ch 15, sec. 4, pp. 668-672
[8] H Gouraud. "Continuous Shading of Curved Surfaces.," IEEE Trans. Comput, vol. c-20, no. 6, Jun. 1971, pp 623-629.
[9] P. Bui-Tuong. "Illumination for computer generated pictures," Commun. of the ACM, vol. 18, no 6, pp 311-317., Jun. 1975.
[10] J. Blinn. "Texture and reflection in computer generated images," Commun. of the ACM, vol. 19, no 10, pp. 542-547, Oct. 1976.
[11] G. Smith. (1996). Asphyxia VGA Demo Trainer #21. Available FTP: ftp.scene.org. Directory: mirrors/hornet/code/tutors/denthor. File: tut21.zip
[12] HELiX. (1997). 2d bump mapping. Available FTP: ftp.scene.org. Directory: /mirrors/hornet/code/effects/bump. File: bumpsrc.zip
[13] J. Blinn. "Simulation of wrinkled surfaces," in Proc. SIGGRAPH '78., Atlanta, GA, 1978, pp 286-292.
[14] A. Appel."Some techniques for shading machine renderings of solids," in Proc. AFIPS '68 (Spring)., San Francisco., CA, 1968, pp 37-45.
[15] T. Whitted, "An improved illumination model for shaded display," in Proc. SIGGRAPH '79, Chicago, IL, 1979, pp 14.
[16] C. Goral. "Modeling the interaction of light between diffuse surfaces," in Proc. SIGGRAPH '84., Minneapolis, MN, 1984, pp 213 – 222.