# RealiBench

## RealiMation 3D Benchmarking Program

***As Used by Ziff-Davis!***

## Description

This benchmark program allows any database to be used. It generates successive frames with a fixed time delta, which guarantees that the *exact* same picture will be generated each time. You can be sure that, no matter what the speed of your cards are, each one will be drawing exactly the same number, size, and shape polygons. (See later on for a full description of the benchmark accuracy). You can vary the length of the test by setting the number of frames the test loops over (see below).

The benchmarking program exe is in the RealiMation BIN directory, and is called "bench.exe". The source code is provided to users of the VSG Developer Edition of RealiMation.

The RealiBench is used by Ziff-Davis for their new generation of 3D benchmarks wth which to compare graphics systems. The December 1996 edition of PC Magazine contains a detailed look at the benchmark.

## Usage

Run BENCH.EXE from the Realimation\bin directory (or copy it somewhere where all the relevent DLL's are).

The program expects as its argument a configuration file. This means that you can set up multiple config files to test different scenarios.
E.g.
bench bench.ini
bench d3d320.ini
bench d3d640.ini
will all run different tests, based on the contents of the INI file.

The initialisation file has the following entries, as shown by the sample below. Just like ordinary Windows INI files, a semicolon at the beginning of a line acts as a comment.

```
[Data]
Results = result.txt
;RealiBase = d:\realibases\misc\eggtimer2.rbs
RealiBase = d:\realibases\newdemo\helisim2.rbs
Driver = rgdd3d4
Frames = 500
Hres = 640
Vres = 480

[Settings]
bFog = 1
bBilinear   = 1
bMipMapping = 1
bTrilinear  = 0
bPerspective = 1
bTextures    = 1
```

**Description of each entry:**

1. Results

This is a string that specifies a file that the results are written to. Note that results are appended to any others already in the file. The results print out the RealiBase used, the driver used, resolution, settings, etc. Run it to see! If you do not supply a file, then the results will go to the DOS screen.

2. RealiBase

This is the filename of the RealiBase to be used in the benchmark.

3. Driver

This is the display driver DLL that is used, minus the ".DLL" extension. You may need the fully qualified pathname if not in the local directory or on the search path.

4. Frames

Lets you change the number of frames over which the test runs. See notes above.

5. Hres & Vres

Specifies the 3D view resolution

6. bFog

If 1, then fog attribute is turned on. The fogging just uses the parameter as set up in the RealiBase, so you can edit using the STE if you like.

7. bBilinear

If 1, enables bilinear interpolation of all textures

8. bMipMapping

If 1, enables mip mapping of all textures

9. bTrilinear

If 1, enables trilinear mipmapping of all textures. NOTE: Unlikely to be implemented on most Direct3D devices, so you will probably not see any difference.

10. bPerspective

If 1, turns on perspective correction of all textures.

11. bTextures

If 1, turns on display of all textures.

## Advice

Set up a number of different ini files, so you can have one, say, for the eggtimer with textures on, fog off, and 200x200 res. You can have another INI file that uses HELISIM2.RBS, with 640x480, and lasts for 5000 frames. You can then just run the benchmark program with different INI files on the command line. We have supplied a sample file BENCH.INI that you can use straight away.

If you want a database that does intense lighting, try LIGHTCUBE1.RBS or LIGHTCUBE2.RBS. The various fountain RealiBases might help too. All of these are in the NEWDEMO or SAMPLES41 directory on the CD. Don't forget that since you have the STE, you can make your own specialsed databases.

## Why the Benchmark is accurate

Here is a description of the time values and how the benchmark used them...

Motion in RealiMation is parameterised by time, so that you can specify that an object is at

position P1 at time t=0, and position P2 at time t=10. By convention, time is in seconds. The RealiMation system interpolates the motion between these P1 and P2 over the time interval.

By default, RealiMation applications will normally generate the time value to feed into the motion interpolator from the computer's realtime clock. So, no matter how fast your card actually generates each picture, the object will ALWAYS be at P2 after 10 seconds of real time. The only difference between a fast graphics display and a slow one, is that the fast display will show smoother movement.

Another way of looking at it is that graphics card X might take, say, 1 second to generate each frame. So, over the 10 second animation, the system will render a total of 10 frames. Graphics card Y, however, may be fast enough to render each frame on 0.5 seconds. So, over the 10 motion interval, it will generate 20 frames. In both these two cases, the object will still move from P1 to P2 over 10 seconds (real elapsed time), but the faster card Y will show smoother movement.

This is not good for benchmarking, since you need to guarantee that each card is being asked to draw exactly the same polygons, in the same location, size, lighting etc, each frame. The solution is, instead of feeding the time from a real clock, the benchmark program generates each frame with a known time value. Typically, it will add a fixed delta to the existing time. In code terms, you can see the difference as follows:

Using RealTime Clock:

```
float tStart = RMClock ();  // RMClock returns current system time in seconds
for (i = 0; i < nFrames; i++)
{
  RTSetViewTime (ViewID, RMClock() - tStart);  // Set the time value for the view
  RTDisplayView (ViewID);     // Generate the polygons
  RTSwapPage    (ChannelID);  // Show the polygons on the display surface
}
```

Using a fixed time delta instead, however, the main loop goes to:

```
float t = 0.0F;  // RMClock returns current system time in seconds
float tDelta = 0.1;  // Time delta is 1/10th second
for (i = 0; i < nFrames; i++)
{
  RTSetViewTime (ViewID, t);  // Set the time value for the view
  RTDisplayView (ViewID);     // Generate the polygons
  RTSwapPage    (ChannelID);  // Show the polygons on the display surface
  t += tDelta;                // Increment time delta
}
```

The upshot of this is that on each occasion the program is run, the SAME time values are used for the same displayed frames, no matter how long they took in "real" time. This makes your benchmark behave the same with all types of graphics card.

## Why can't I have Interaction with the Benchmark while running?

The problem with this approach is that it breaks the guaranteed consistency between runs as explained above.