# RealiBench

## RealiMation 3D Benchmarking Program

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

Last Updated : 22nd January 1997

## Description

This benchmark program allows any RealiBase to be used. It generates successive frames with a fixed time delta, to guarantee that the *exact* same picture will be generated each time. You can be sure that, no matter what the speed of your system and graphics cards, 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 "RealiBench.exe" (An icon is created for the program in the RealiMation program group). 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 to compare graphics systems. The December 1996 edition of PC Magazine contains a detailed look at the benchmark.

The resulting **RealiMark** is an average frame rate (frames per second), and so the bigger the number the better.

## System Requirements

RealiBench will work on Windows 95 and Windows NT (both 3.51 and 4.0) operating systems.

The only other software you must have installed is either RealiView or one of the RealiMation Space Time Editor (STE) versions. If RealiBench was installed as part of an STE, then you should just be able to run it (see Usage below).

If you just downloaded RealiBench from Datapath's website (www.realimation.com), you may also need to download and install RealiView. Once you have done this, simply copy the RealiBench files into the BIN subdirectory of the RealiView program.

## RealiBases

RealiBench can work with any RealiBase that describes a 3D scene with views, shapes, actions, textures and any other information required to describe a full motion 3D world. If you have a RealiMation CD (either from Datapath or as part of a bundle deal), then the SAMPLES directory contains a large number of example RealiBases you can use. A set of good ones are in the HELISIM subdirectory, but you can use any of the others.

Users should note that turning textures on and off as part of RealiBench is dependent on a RealiBase having textures as part of the scene. If no textures exist, RealiBench will still run through all the tests as if textures were actually there. For this reason, we recommend that RealiBench only be used with textured scenes.

The type of scene you use may also be of interest to you. If you are wanting RealiMation for an architectural type application, then it makes sense to use a RealiBase that suits your needs, rather than say a Newtons Cradle object.

## Usage

Double click the RealiBench icon.

The program is a dialog based application that can take initialisation files containing RealiBases to use, settings, display sizes, display drivers etc. A sample "BENCH.INI" file is supplied. On loading this, you may need to change the RealiBase to one installed on your system.

Once you have set up the data, you can run either a single test using the current settings, or a more comprehensive test that executes each checkbox (see below) in turn building up more and more overhead. Note that if a display driver is not capable of a particular feature (e.g. not all drivers can do mip mapping or anti-aliasing), these tests are not executed and a "N/A" entry ("Not Available") is created in the results.

You can save your settings at any time by selecting the Save button. Note that you should ensure the file is saved with a ".INI" extension.

On completion, a message box displays the results, with an option to save these results to a file. If you are just running a single benchmark, the results can only be written to a "human readable" file. You can append results of successive runs to the same file, and add comments if necessary.

The "Run through all options" benchmark results can be saved to an ASCII text file formatted for import into a spreadsheet such as Excel. In this case you can set what each row means, and if you run this several times, you can accumulate results over different display drivers, RealiBases, or resolution settings, etc. When accumulating results in the same file, you should not change what each row means, otherwise the spreadsheet you get will be meaningless. Try this feature and look at the spreadsheets to give yourself a better idea as to how results are generated.

To display your results to a spreadsheet, we have supplied a Microsoft Excel spreadsheet complete with a formatting macro to simplify the task of creating a nicely formatted table and a results graph. A shortcut called "RealiBench Results" is created in your RealiMation program group. Follow the instructions as they appear.

**Description of Main Dialog:**

Load/Save Config Buttons
Loads and saves a RealiBench configuration file. These should have a ".INI" extension.

Driver & Change Button
This displays the current display driver, and a button allows you to change it. The first time you run RealiBench and select this button, you will be asked which drivers you wish to have available. Check all of the boxes so you can change drivers later.
NOTE: If you want to force RealiBench to look again for its drivers (maybe to update the list), edit your registry to delete the following key:
    //HKEY_CURRENT_USER/Software/RealiBench/Channel
See the STE help file for details on editting your registry (search on "registry");

RealiBase and Change RBS Button
This is the filename of the RealiBase to be used in the benchmark, and the button allows you to change it.

View Width and View Height
Specifies the 3D view resolution that the benchmark will run in. Typical values are 320x200, 640x480, etc. It is useful to change this to see how different display technologies cope with increasing 3D display area.

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

benchmark.
Generally, you should have a sufficient number of frames so that the duration of the benchmark is longer that just a few seconds to minimize slight timer innacuracies on certain computers. We recommend a value of at least 200 frames, but 1000 is even better especially when measuring very fast systems or simple RealiBases.

Check Boxes
*Textures*: Turns off all textures in a scene, and runs the RealiBase in the view's display mode, as set up by the STE (normally smooth (Gouraud) shading).
*Bilinear* : Enables bilinear interpolation of all textures
*Mip Mapping* : Enables mip mapping of all textures
*Mip Map Blending* : Enables trilinear mipmapping of all textures when used in conjunction with the Mip Mapping flag.

NOTE: for a full description of these features, see the STE help file.

Run Benchmark Button
Pressing this runs a single benchmark with the settings in the dialog.

Run Through All Options Button
Pressing this runs several benchmarks, each turning on the available checkboxes. This is probably the most generally useful benchmark.

## 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.

## Notes on Certain Display Drivers

1. Capability Deception

You should be aware that RealiBench will only run those tests that the underlying display driver reports are available. For example, the mip mapping test will only happen if the display driver is capable of mip mapping.

There is scope here for display driver "abuse" of benchmarks. For example, when running the OpenGL driver, mip mapping and mip map blending (aka trilinear filtering) are reported as being available. Certain graphics cards, however, while reporting to the operating system that they can do these things, actually ignore the request, and so do not do the mip mapping.

For example, the GLINT family of cards explicitly disables the mip mapping, and instead just runs the 3D view with bilinear filtering, and so reports good results to the benchmark when, in fact, the RealiMark should be zero.

To guard against this, you should check the documented capabilities of the graphics card. if you have any doubts, then please contact Datapath Ltd, who will be pleased to offer you their knowledge of different graphics cards and chipsets.

You can detect that this capability deception is happening by looking at the benchmark results. If the RealiMarks change only very slightly betwen features, then there *may* be a problem, and you should investigate further.

2. Vertical Refresh Rates

Another issue to be aware of is that some 3D graphics cards can create pictures faster than the vertical refresh rate of the monitor. In general, these drivers will appear to never run faster than

the screen refresh rate. In this situation, RealiBench will be showing results that are slower than the card is actually capable of doing.

An example here are cards based on the 3Dfx chipset (e.g. 3Dfx Interactive Obsidian Pro, Diamond Monster 3D, Orchid Righteous 3D). You can tell these cards to ignore vertical synchronisation, in which case you will get true results. See the Display Drivers Readme file for details on how to do this.

3. Corrupt Textures with OpenGL

RealiBench makes use of Texture Objects when driving OpenGL. Some OpenGL accelerator cards do not always implement these correctly, however, giving the effect of textures placed on the wrong parts of objects. RealiBench can force the OpenGL driver to completely reload the texture information between each benchmarking run to get around this problem. To do this, edit the registry, and create a new DWORD value:

//HKEY_CURRENT_USER/Software/Realibench/Settings/ReloadImages

Setting the value to 1 will force realiBench to reload textures. The default value is zero, which does not do this.

NB: See RealiMation STE help file for registry editting information (search on "registry");

## 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.

## Copyright Information