Summary

Changes in Release 11

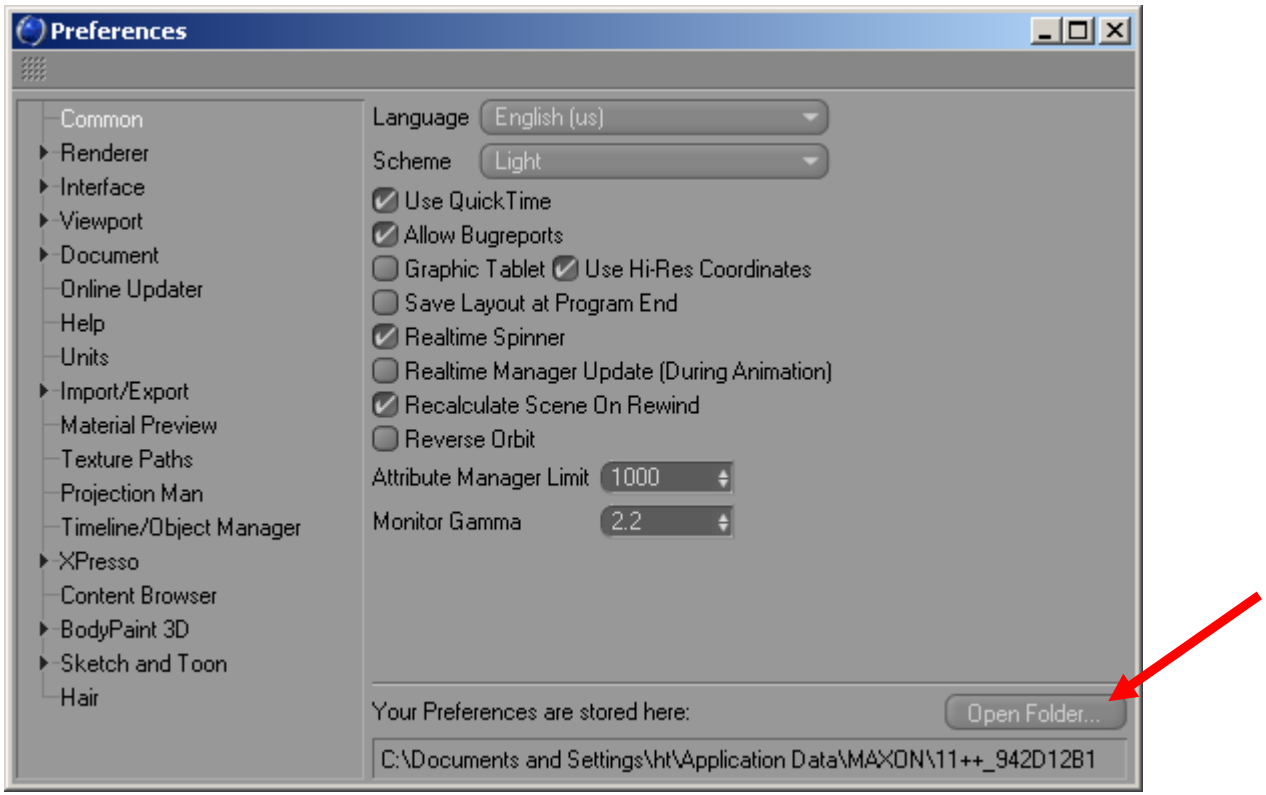
- C4D application folder can‘t be modified anymore

  (with the exception of „plugins“)

Changes in Release 11

- User settings and preferences are stored in a different location

Changes in Release 11



Changes in Release 11

- Save plugin preferences using GeGetStartupWritePath()
- Data can’t be written into your plugin directory anymore (Vista/Leopard changes)
- C4D installation into application folder is not recommended for developers

API Changes in Release 11

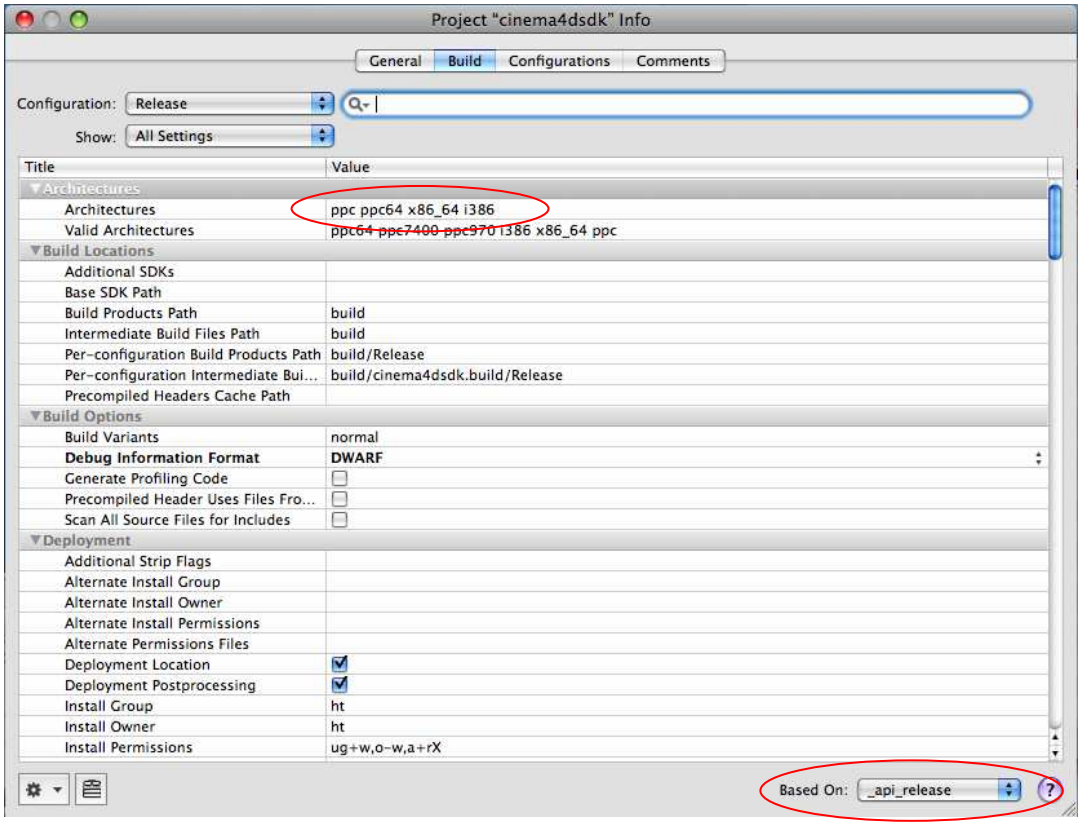
- Updated project settings for XCode 3.1 on OS X

- OS X 10.5 necessary for development
  (Executable will run under OS X 10.4 as well)

- Support for 64 Bit on OS X
  (C4D defaults to 32 Bit for now - this will change soon)

API Changes in Release 11

- If your project is based on the C4D API settings, no modifications are necessary



API Changes in Release 11

- Datatype WORD renamed to SWORD
  (due to conflicts with Windows API)

  Advantage of using C4D‘s datatypes: under all compiler systems on all platforms you get the same exactly defined behaviour
  (e.g. datatype ‚long‘ different on PC and MAC 64-Bit)

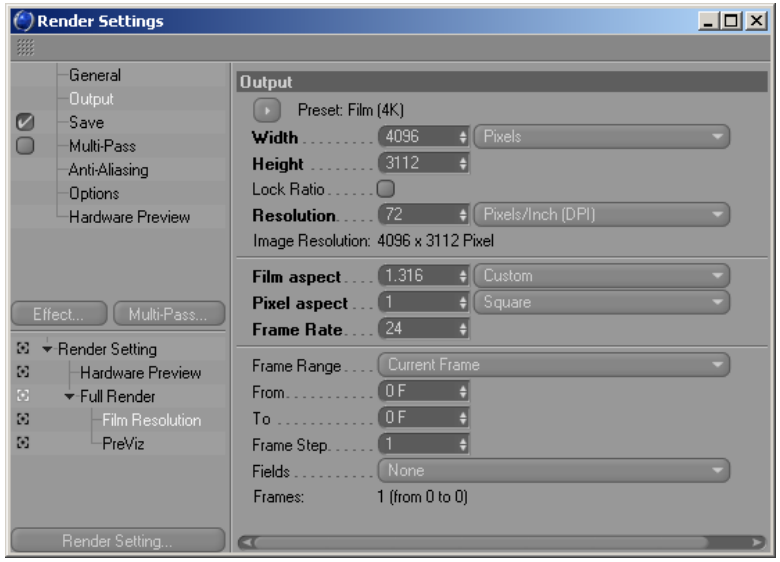
- No more support for Visual Studio 6

API Changes in Release 11

- STL overloads for delete added
- For future releases, new / delete overloads will be removed
- Make sure to use gNew/gDelete for C4D Datatypes

API Changes in Release 11

- Access to new Render Settings




```
class MultipassObject : public BaseList2D
{
  private:
    MultipassObject();
    ~MultipassObject();
  public:
    MultipassObject *GetNext(void) { return (MultipassObject*)AtCall(GetNext)(); }
    MultipassObject *GetPred(void) { return (MultipassObject*)AtCall(GetPred)(); }
};

class RenderData : public BaseList2D
{
  private:
    RenderData();
    ~RenderData();
  public:
    RenderData *GetNext(void) { return (RenderData*)AtCall(GetNext)(); }
    RenderData *GetPred(void) { return (RenderData*)AtCall(GetPred)(); }
    RenderData *GetDown    (void) { return (RenderData*)AtCall(GetDown)(); }
    RenderData *GetUp      (void) { return (RenderData*)AtCall(GetUp)(); }
    RenderData *GetDownLast(void) { return (RenderData*)AtCall(GetDownLast)(); }

    PluginVideoPost *GetFirstVideoPost();
    void InsertVideoPost(PluginVideoPost *pvp, PluginVideoPost *pred=NULL);
    void InsertVideoPostLast(PluginVideoPost *pvp);

    MultipassObject *GetFirstMultipass();
    void InsertMultipass(MultipassObject *obj, MultipassObject *pred=NULL);

    static RenderData *Alloc(void);
    static void Free(RenderData *&v);
};
```

- Render Setting based on Descriptions
- PLUGINFLAG_VIDEOPOST_ISRENDERER

Licensing Changes

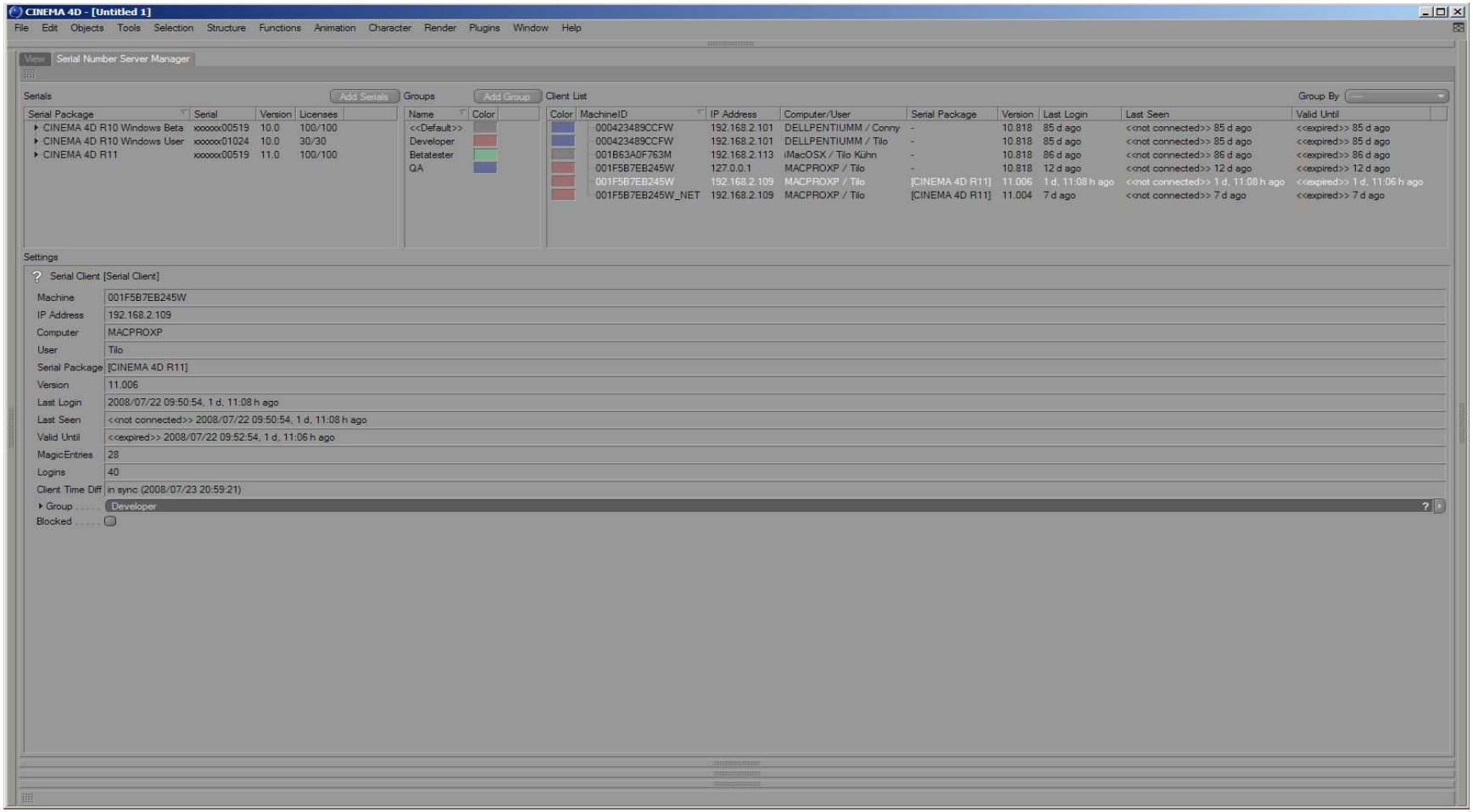
- New Savable Demo version
- New License Server

Savable Demo version

- GeGetVersionType() returns VERSION_SAVABLEDEMO and VERSION_SAVABLEDEMO_ACTIVE

- Runtime is 42 days
- No limitations except for render resolution restricted to 640x400 and no NET
- No serial number available

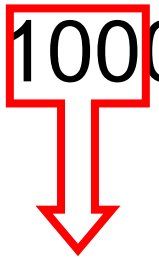
License Server



License Server

- GeGetSerialInfo(SERIAL_MULTILICENSE) returns multi-license or empty string (regular C4D version)

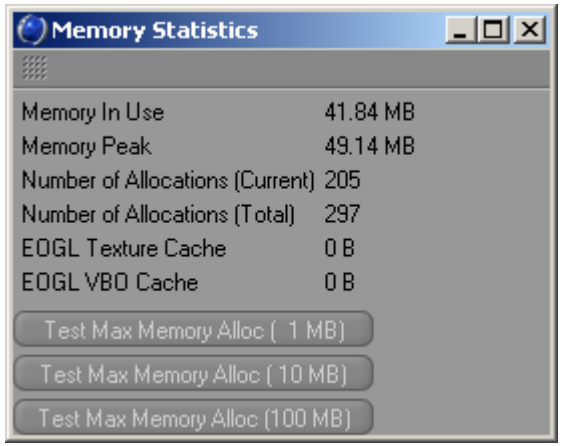
- Multi-License Structure
  20110000519-ABCDEF



Number of purchased licenses

Stability and Testing

- c4d_debug.txt to find memory leaks (place in user directory)
- C4D SDK Memory Statistics Dialog to see realtime use of memory



Stability and Testing

- Use of constants to avoid compatibility problems

```
BaseContainer *bc = obj->GetDataInstance();
if (!bc) goto error;

bc->SetLong(1001,1.0);
```

Wrong!

```
#include "../../advanced render/res/description/Xsss.h"

BaseContainer *bc = obj->GetDataInstance();
if (!bc) goto error;

bc->SetLong(SSSSHADER_STRENGTH,1.0);
```

Right!

Stability and Testing

- Crashes due to use of global static classes

```
AutoAlloc<String> my_global_string1;
String my_global_string2;
Filename my_global_filename;
```

Wrong!

- Crashes happen randomly and are often hard to reproduce
- Allowed are elementary data types and data types that don‘t need to be allocated (LONG, Real, CHAR, GE_SPINLOCK etc.)
- Any other SDK data type cannot be placed in the global scope

Stability and Testing

- Crashes due to wrong cleanup code
  - Crashes happen randomly and are often hard to reproduce
  - PluginEnd() vs. C4DPL_ENDACTIVITY
  - Use of C4DMSG_PRIORITY

Stability and Testing

- PluginEnd() vs. C4DPL_ENDACTIVITY

```
}

void PluginEnd(void)
{
  BaseBitmap::Free(g_pNoisePreview);
  gDelete(g_pbMenuContainer);
  Semaphore::Free(g_pBakeSemaphore);
  g_NoiseContainer.FreeContainers();
  FreeNoisePreviews();
}

Bool PluginMessage(LONG id, void *data)
{
  switch (id)
  {
    case C4DMSG_PRIORITY:
      SetPluginPriority(data, C4DPL_INIT_PRIORITY_SLA);
      return TRUE;

    case C4DPL_ENDACTIVITY:
      FreeBaker();
      return TRUE;
  }

  return FALSE;
}
```

Low-Level cleanup
-e.g. Strings, Filenames
-unloading of imported DLLs

Be careful with BaseContainers that contain CustomDataTypes!

High-Level cleanup
-e.g. Dialogs, BaseContainers, Datatypes
-Finishing of all disk activity, preferences storage etc.
-Termination of any running Tasks!

Stability and Testing

- C4DMSG_PRIORITY

  (Definition of Startup / Cleanup order)

```
Bool PluginMessage(LONG id, void *data)
{
  switch (id)
  {
    case C4DMSG_PRIORITY:
      SetPluginPriority(data, C4DPL_INIT_PRIORITY_SLA);
      return TRUE;
```

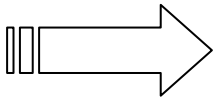
```
// priorities
#define C4DPL_INIT_PRIORITY_XTENSIONS       20000
#define C4DPL_INIT_PRIORITY_OBJECTS         19000
#define C4DPL_INIT_PRIORITY_MODELING        18500
#define C4DPL_INIT_PRIORITY_SHADER          18000
#define C4DPL_INIT_PRIORITY_ADVANCEDRENDER  17000
#define C4DPL_INIT_PRIORITY_MOCCA           15000
#define C4DPL_INIT_PRIORITY_NEWMAN          12000
#define C4DPL_INIT_PRIORITY_SLA               850

#define C4DPL_INIT_PRIORITY_MODULES         10000
#define C4DPL_INIT_PRIORITY_PLUGINS             0
```

Stability and Testing

- Most dangerous places for hooks are:

  UndoHook
  BackgroundHandler
  VideoPostData with VIDEOPOST_INHERENT
  MessageData
  SceneHookData
  overloaded ::Read/::Write routines

All of those can greatly affect the application‘s stability and lead to data loss for the end user!



Be extremely careful, use defensive coding style and take the time to properly test when working on those areas

Stability and Testing

- This Virtual Function Call will crash:

```
PluginVideoPost *node;
for (node = renderdata->GetFirstVideoPost(); node; node=node->GetNext())
{
  VideoPostData *pVPData = (VideoPostData*=)node->GetNodeData();

  LONG info = pVPData->GetRenderInfo();
}
```

Wrong!

- Beware of calling NodeData members!
- pVPData can be NULL, check necessary
- A plugin is not allowed to use the virtual function table of another plugin! This will only work with the same plugin, not over plugin boundaries
- To avoid crashes call above must be replaced by

```
PluginVideoPost *node;
for (node = renderdata->GetFirstVideoPost(); node; node=node->GetNext())
{
  VideoPostData *pVPData = (VideoPostData*=)node->GetNodeData();
  if (!pVPData) continue;

  LONG info =
  ((pVPData->*((VIDEOPOSTPLUGIN*)C4DOS.Bl->RetrieveTableX(pVPData,0))->GetRenderInfo)(node);
}
```

Right!

Stability and Testing

All parts of the execution/drawing pipeline of CINEMA 4D are threaded. Multiple renders can run at the same time.

This means that all calls to TagData::Draw(), ShaderData::Draw(), SceneHookData::Draw(), SceneHookData::Execute(), TagData::Execute(), ObjectData::GetVirtualObjects(), ObjectData::Draw() etc. are made from a thread.

Other threads (like the manager redraws) access this data at the same time. That's why no scene modifications of **any** kind must be made in those routines. For example inserting an object into the scene is enough to trigger a crash!

As an exception, modifications are allowed that change the object's parameters like position, or anything set through SetDParameter(). However this should only be done by tags (expressions) and not by generator objects.

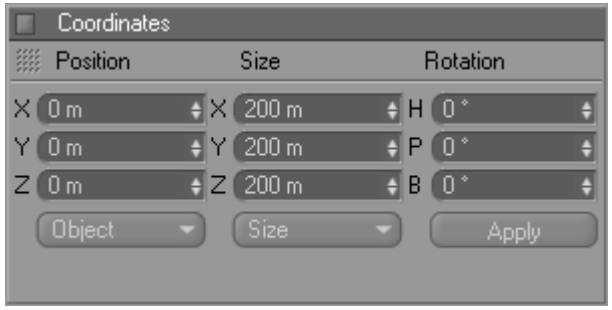
GetVirtualObjects() is allowed to do ANY modification within its own cache.

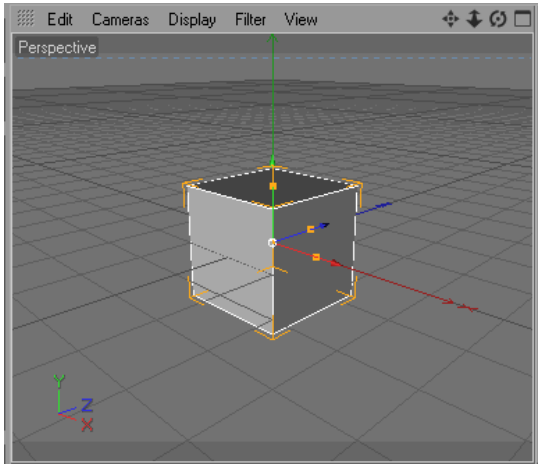
Threading (from the SDK documentation)

Stability and Testing

```
void MyObjectData::GetDimension(BaseObject *op, Vector *mp, Vector *rad)
{
     BaseObject *cop = op->GetCache();
     if (!op) return;

     *mp  = cop->GetMp();
     *rad = cop->GetRad();
}
```

Wrong!





This code will crash as the viewport thread creates the cache while the coordinate manager retrieves the dimensions, accessing the cache that is being written to at the same time.

Stability and Testing

Preventing other threads from working

- Any modal dialog calls StopAllThreads() ahead of time (but redraw can be started in the background again – this is important to know for helper threads)
- Call StopAllThreads() before any operation in Non-Modal dialogs (e.g. if user clicks on button)

Stability and Testing

- CurrentStateToObject

  Object duplication necessary as CSTO modifies existing caches

Stability and Testing

- Include OS before C4D headers
- Keep code with shared OS and C4D calls as small as possible and collect in one place
  (better portability & less datatype conflicts)

Stability and Testing

- Any API function may fail! Check for everything, even if it seems obvious
- Most prominent: GeAlloc(), gNew
- Any division needs to be checked for 0.0
- SSE2 changes runtime behaviour: both examples will eventually crash!

```
Real q,v = oldvalue;
if (v!=0.0)
{
  q = 1.0/Sqrt(v);
}
```

```
Real q = v;
if (v!=0.0)
  q=1.0/q;
```