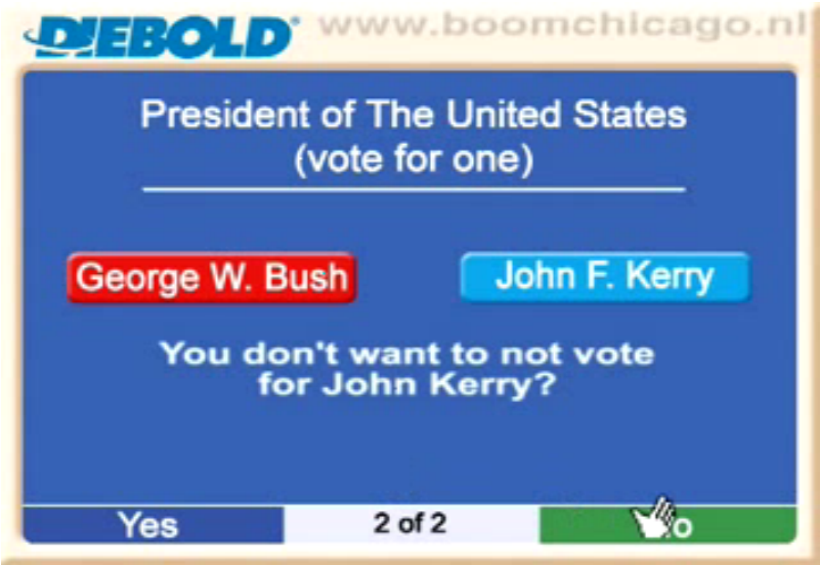
User Interface Design



http://www.youtube.com/watch?v=yUdpj3gJofQ

"There are no bad players, only bad designers"

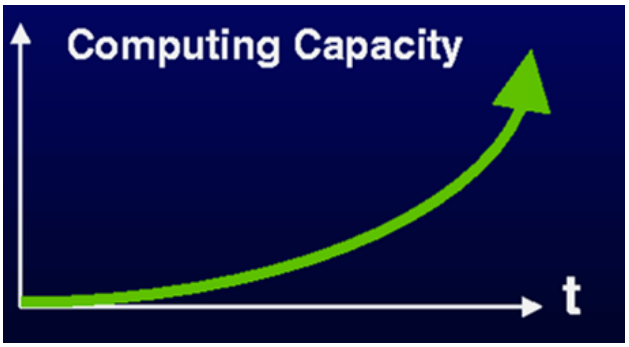
Joshua Nuernberger, game designer:

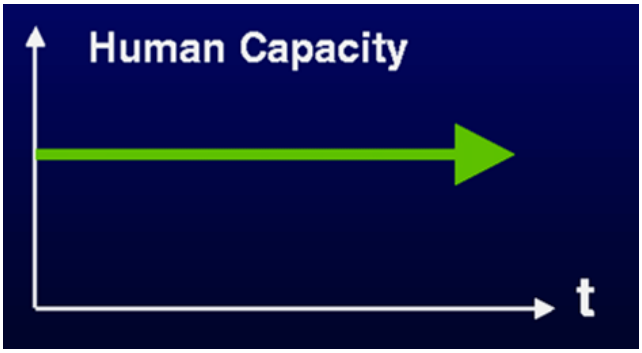
- I want the player to go a certain way, open a certain door, or go down a certain ramp – but what's the obvious direction for me is not the obvious direction for the player.
- This doesn't mean that the player is mentally impaired or 'bad at video games.' There are no bad players – only bad designers

Gemini Rue
http://www.youtube.com/watch?v=Yh2YYDcfT1Y

UI Design is Difficult

- In many modern programs, the user interface code constitutes the bulk of program, i.e., 70-80%
- For the most part, the user interface is the key to the success or failure of a program; the user interface defines the product
- Computer speed increases by Moore's law, human capacity does not





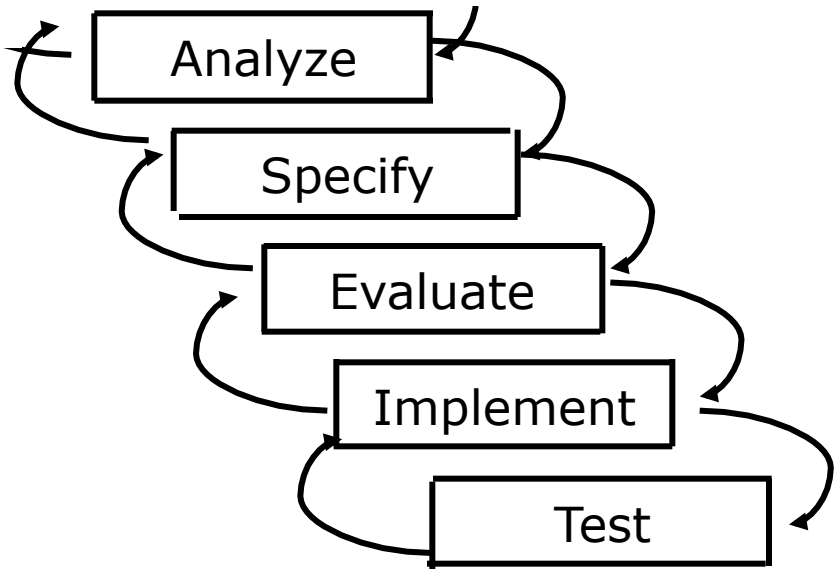
- Creating a good UI is often harder than software engineering because UI design requires much more than software engineering skills

Design Methodology

Process model

- Not a step-by-step approach, or a pipeline; for user interface design, this model is less linear than software engineering models and more complex because of human element

- Model below is not a hierarchy: has feedback loops



1. Analysis

- Learn about and from users
  - ethnographic observation, interviewing
  - site visits; shadow the user
  - become the user
    - The 90/10 rule: 10% of the features will be used by the user 90% of the time

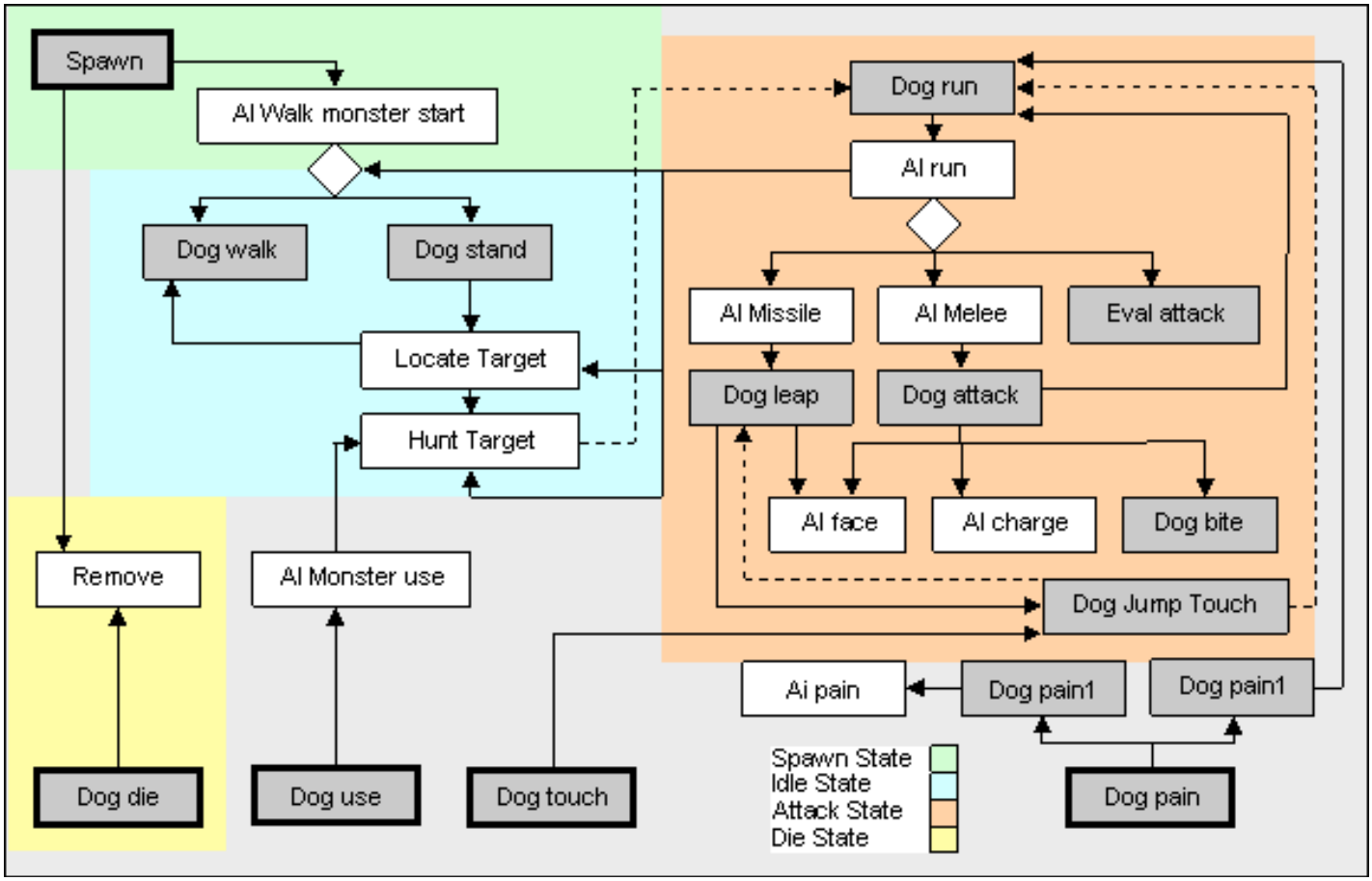
Heavy Rain

Know Thy User

Heavy Rain

2. Specification/Design

- Often helpful to use FSM specification

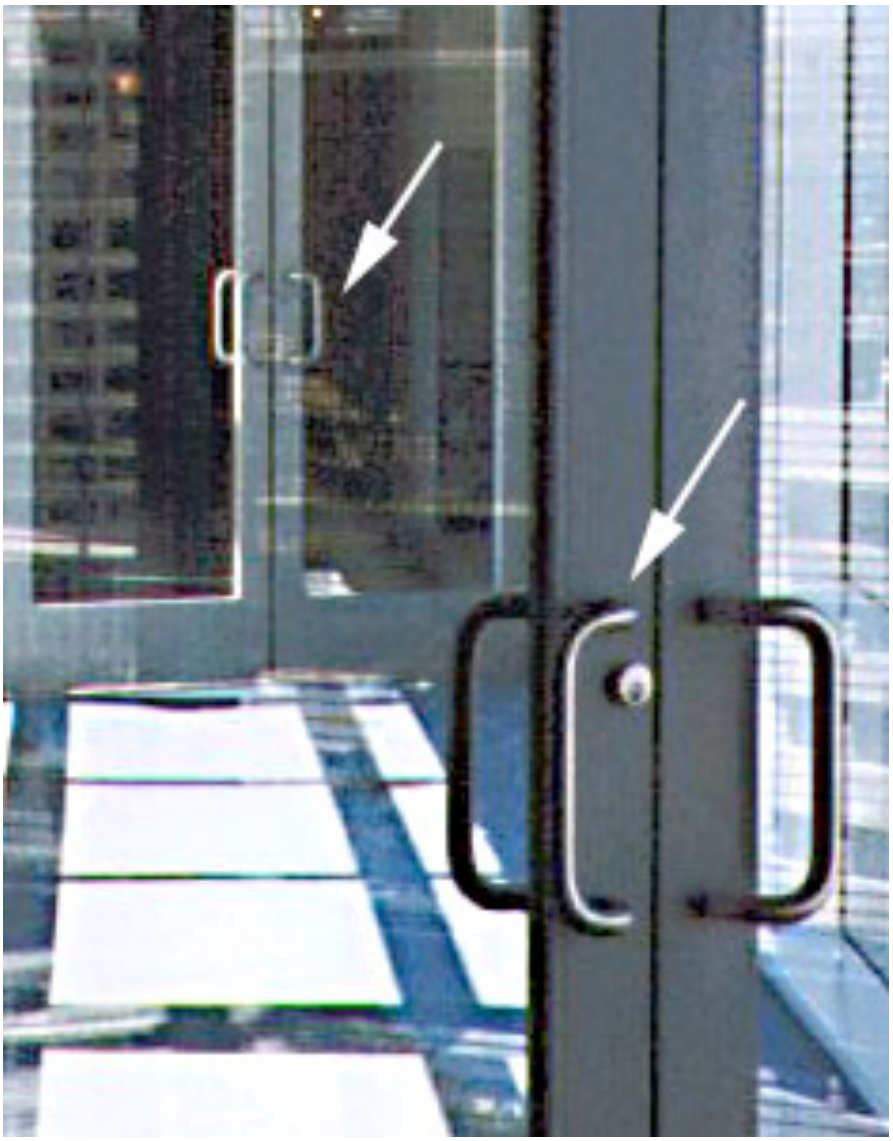


http://ai-depot.com/FiniteStateMachines/FSM-Framework.html

- Then follow with detailed description of design (screenshots, menus, interaction etc)

Design Principles

- The 6 (or 9, or 11) design heuristics ☺



The design of everyday things,
D.A. Norman

Design Principle I: Consistency

Coding consistent with common usage (for a given culture), both in games and general UI

- Red = bad, green = good
- Left = less, right = more
- Colors must always code info in the same way

Devices used same way in all phases of interaction

- "Jump" key is always the same
- Double-press means "higher"
- Function key meanings are consistent throughout application
- Menu placement is consistent

Design Principle II

- Provide feedback
  - A selected object or menu command should be highlighted, so the user knows the action has been accepted
  - When player shoots, bullets should put holes in objects, or at least dents
  - Prompt for next input
  - Indicate the computer is working on the problem if processing the command takes more than 1 or 2 seconds
    - use a dial or a gauge
  - Tell the user that the operation has been completed

Design Principles III-VI

- Minimize error possibilities
  - System should provide user with only those commands that are valid in the current context
    E.g., can't delete when there's nothing to delete; don't copy when nothing is selected
  - Provide visual cues to those actions that are valid/expected in the current context
    E.g., highlight portals or "jumpy" objects to be picked
  - Warn the player/user of the effect of their actions
    A.k.a., "The player must always blame themselves"
- Provide error recovery
  - Undo, Abort/Cancel
- Accommodate multiple skill levels
  - E.g., both menus and kbd shortcuts
- Minimize memorization (e.g, avoid acronyms)

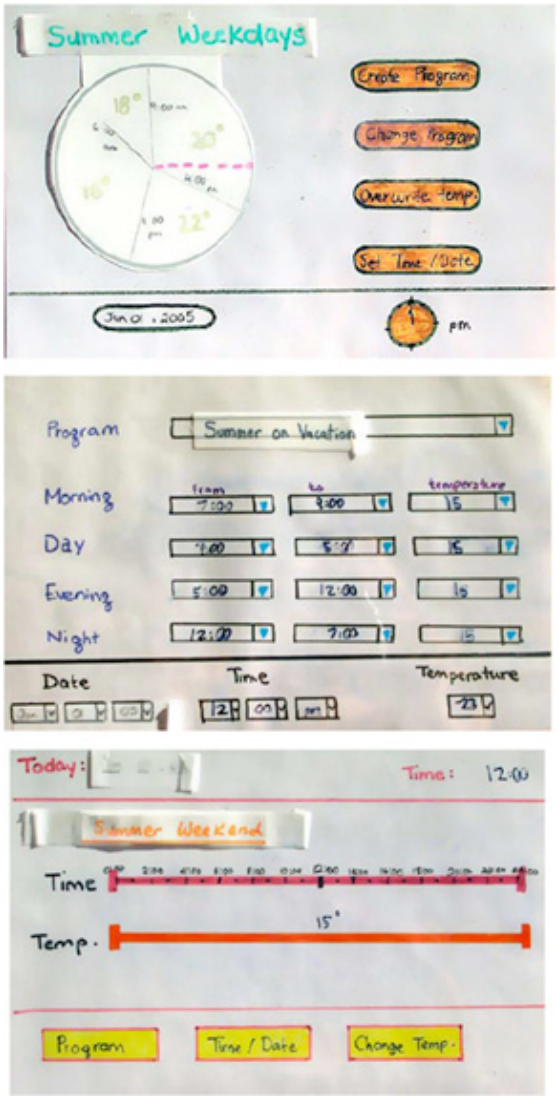
Example: Provide Visual Cues

Gemini Rue
http://www.youtube.com/watch?v=Yh2YYDcfT1Y

3&4. Evaluation & Implementation

Evaluation & Implementation: Use prototyping

- In many circumstances, LoFi prototypes work *better than* HiFi prototypes



- Goal: to try out lots of ideas fast/early
  - "to get a good idea, get lots of ideas"
  - instant iteration
- Users suggest more fundamental changes
  - users focus on high level rather than color, labels, graphical details
- Status is clear, no unreasonable expectations

5. Testing

Involve real users/players

- Both new and experienced users
- Designers or developers do not make good testers
  - "It works fine when I use it."

Designers present at usability testing

- But gagged in background, or at least can't use their hands
- To listen, observe, learn, and sweat!

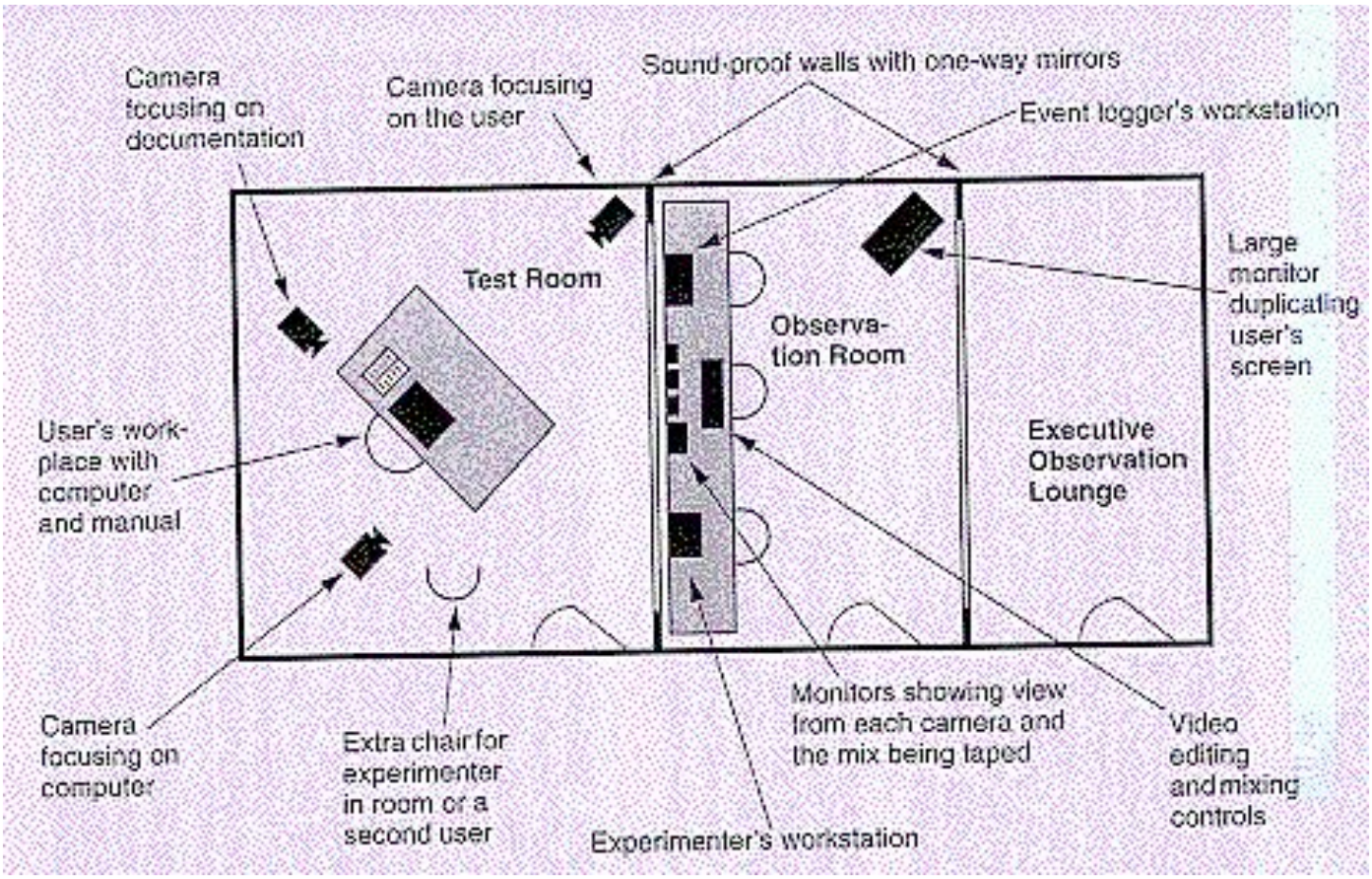
Experimental design methodology (stats-based)

- What to test and how to test it takes ingenuity and experimentation
  - granularity spectrum: individual interaction techniques to task performance

Plan on major investments

- Particularly for commercial products – UI may cost as much as the rest of the system

Usability Lab



http://www.cc.gatech.edu/classes/cs6751_97_winter/Topics/usab-expers/

Parting Words of Wisdom

- OpenGL is not great for buttons and menus
- If you plan to use them, give QT in conjunction with OpenGL a try

- Try text mapped to billboards to explain what your game is about
- Or to show dialog options
- Or to provide hints to the player

- Everything we learned about color perception applies straight-up to UI design (e.g., color blindness, saturation, contrast etc)

- Quality of interaction can really make or break a game

- Be creative!

Summary

- UI is the component most critical to user satisfaction, and probably the largest component of today's interactive apps
- UI design model
- User-centered analysis is mandatory: "Know thy user!"
- Specification via FSM
- 6 Design Principles: consistency, provide feedback, minimize error, error recovery, multiple skill levels, minimize memorization
- Rapid prototyping and usability testing
- UI design: separate field; overlaps with sw engineering, perception studies and art/design; its own discipline and literature

What to NOT Expect

www.insomniacgames.com/blogcast/

Ratchet & Clank: 10 Years of Concept Art

by John Fiorito

10 this year and to celebrate the anniversary Sony will release the *Ratchet & Clank HD*
er since the first Ratchet & Clank game hit the streets fans have written us wanting to see
ncepts are just one of many pieces we used to put our games together, they are a great
ry of the series. So I searched around the studio and pulled together a collection of
way back to 2002 when Insomniac started working on a new game…

2000. We were wrapping up *Spyro: Year of the Dragon* and looking ahead to the
ending a year developing a new game code-named "*I-5*" and later known as "*Girl with a*
cancel the project and shift gears (To find out more about *Girl with a Stick* and its ultimate
Full Moon Show podcast #49 here). That left us with only a few months to present a new
d working on a game set in an alien galaxy that featured lots of crazy gadgets. Right
(about 40 of us) focused on bringing this idea to life. We started to develop our tech,
mation, story, design, sound… everything! On the concept art front, Insomniac character
ted these early sketches of our heroes:

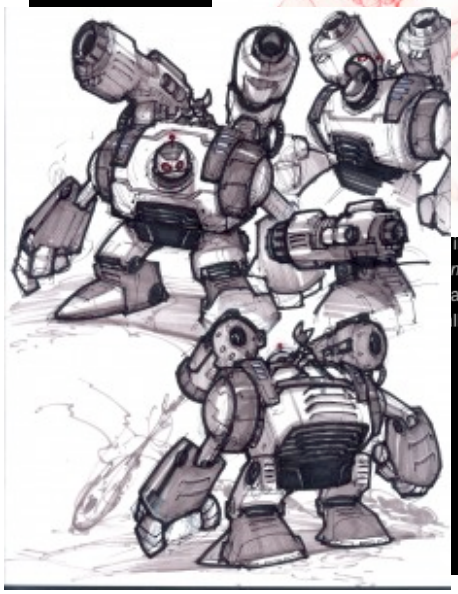
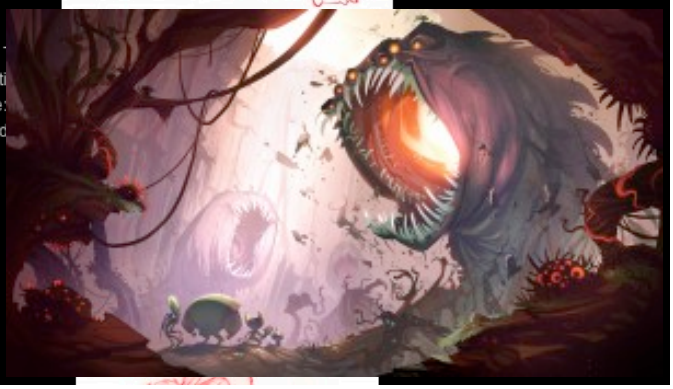
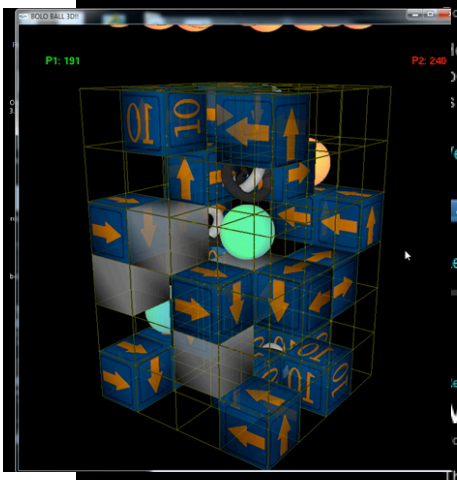


image of Ratchet.
ndo traversal abiliti
acteristics in place
lly his reptilian boc



What to Expect



www.insomniacgames.com/blogcast/

Research & Development

Mike Day: Vector length and normalization difficulties

osted on July 16, 2012

lere's a short account of an easily overlooked difficulty with vector length and vector normalization functions, ogether with one way of solving the problem. We'll use 3-component vectors by way of illustration, but the idea s easily extended to longer or shorter vectors, quaternions, etc. Single-precision floating point is assumed.

'ector-length-and-normalization-difficulties.pdf

Like 1 +1

eave a comment

esearch & Development

Mike Day: Extracting Euler Angles from a Rotation Matrix

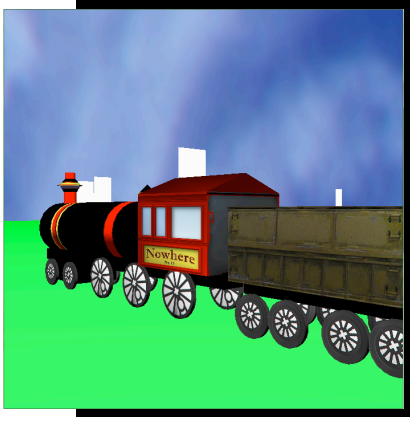
osted on July 11, 2012

This article attempts to fix a problem which came up when implementing Ken Shoemake's Euler angle extraction in the context of a single-precision floating point library. The original Shoemake code uses double precision, which presumably maintains sufficient precision for the problem not to arise.

euler-angles.pdf

Like 2 +1

Leave a comment



Research & Development

Mike Day: Overlap test for spotlight cone vs sphere

Posted on July 3, 2012

A 'spotlight cone' can be pictured as an ordinary right circular cone, the base of which has been inflated outwards to form part of a spherical surface centred on the cone's apex. This shape is useful in lighting systems for finding all the objects which could be influenced by a given spotlight. This document shows a stripped-down calculation for testing whether a given spotlight cone overlaps a sphere, typically the bounding sphere of an object which may be lit. The calculation can be used as the basis for an optimized SIMD test for checking a batch of several spheres against a given spotlight cone.

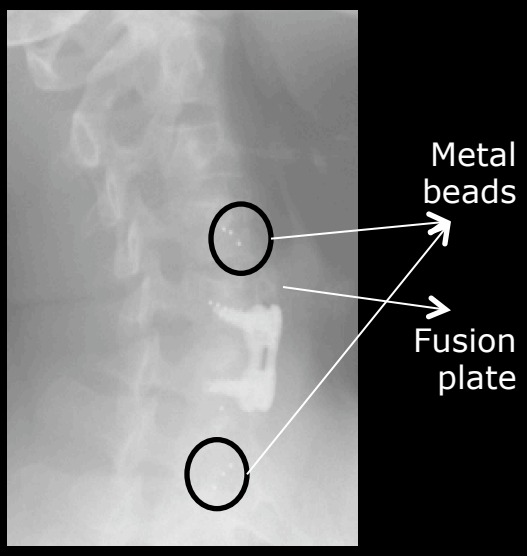
spotlight-cone-vs-sphere.pdf

Like 1 +1

Leave a comment

Each Day: 1,128 Spinal Fusion Surgeries

J. Merriman, Pittsburgh Tribune-Review 2009



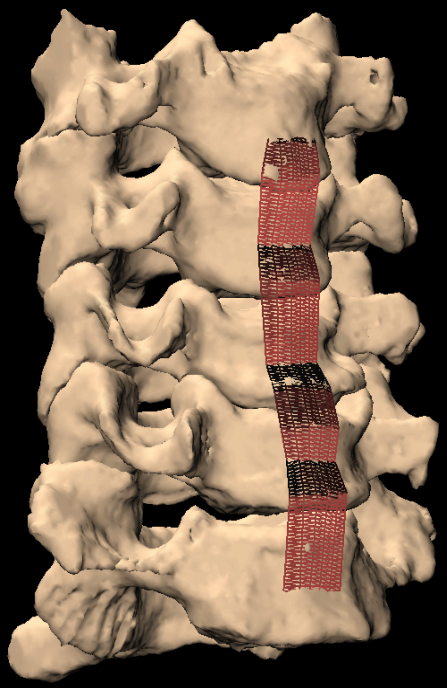
From Numbers to Insight

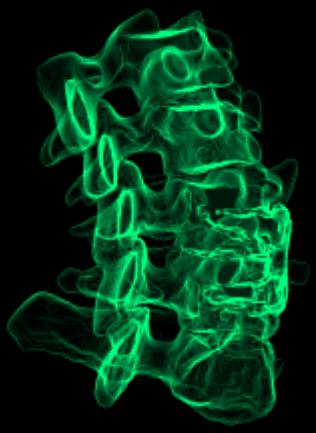
R. Hamming, 1962

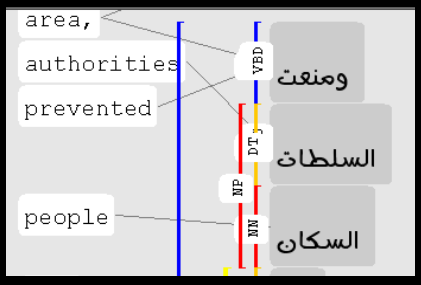
Computer graphics and visualization help us

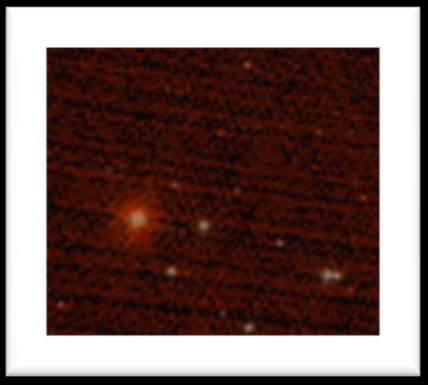
- model complex systems
- make predictions about their behavior
- harness the power of our visual perception system

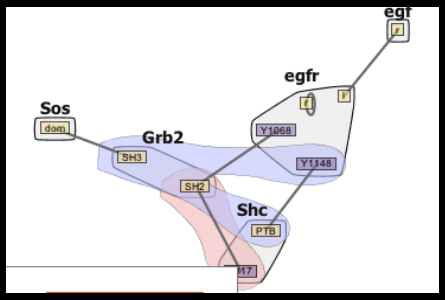
to make *insights* into complex processes possible.

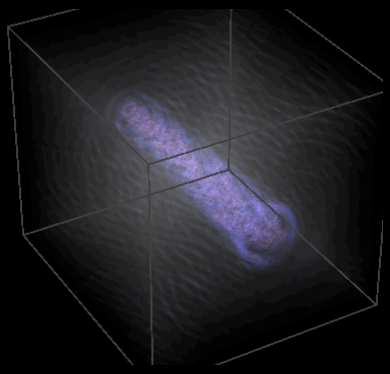


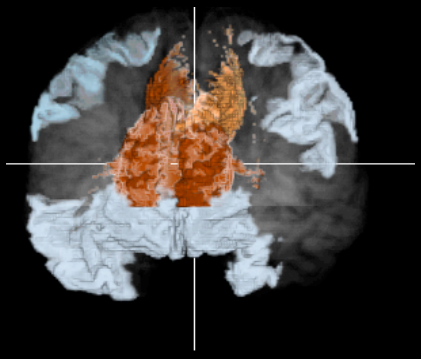


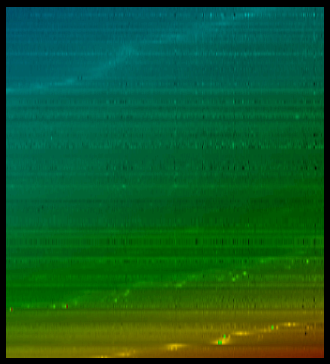

area,
authorities
prevented
people
VBD
DTJ
NP
NN
ومنعت
السلطات
السكان




egf
egfr
Sos
dom
Grb2
SH3
Y1068
Y1148
SH2
Shc
PTB







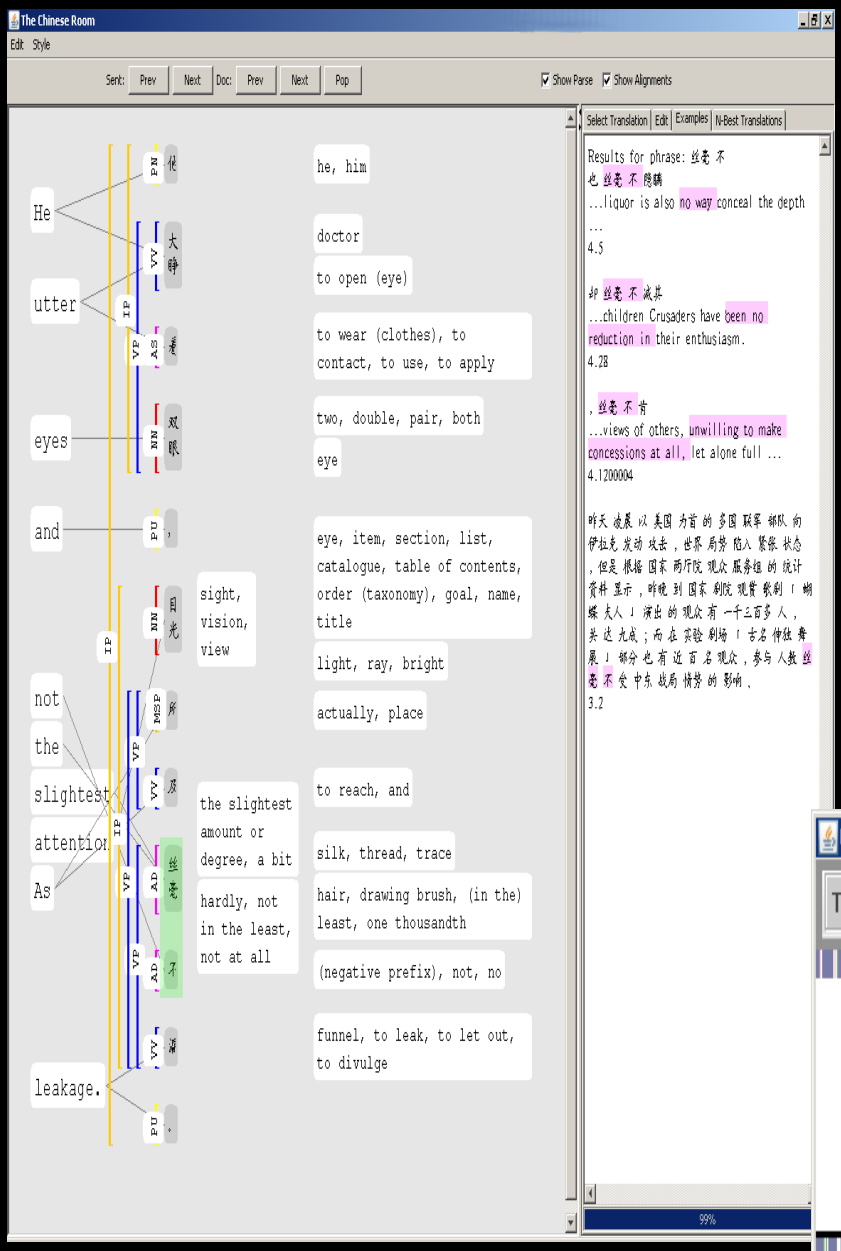
What does this say?

他 大睁 着 双眼， 目光 所 及 丝毫 不 漏 。

Machine translations:

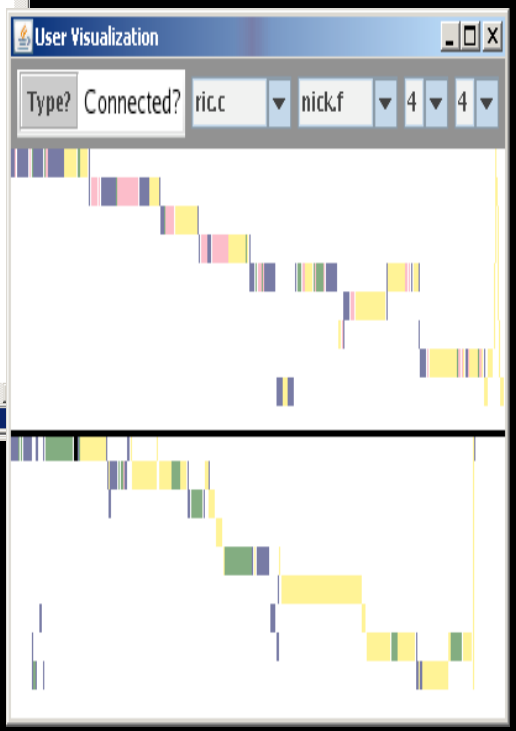
- “He utter eyes and not the slightest attention As leakage.”
- “He Zhengzhao eyes, eyes can no leakage.”

The Chinese Room



MT: "He utter eyes and not the slightest attention As leakage."

Chinese Room MT: "His eyes were placed wide-apart; nothing escaped their attention."



Bonus: Drawing Efficiently

- The drawing process breaks down as follows.
  - Configure our environment/set up a context
  - Configure the render state/matrices
  - Give GL vertex information to render

- There are several ways to pass vertex information to GL
  - We'll look at some of them

- Vertices consist of at least a position, but may also contain normals, color information, and texture coordinates
- We can create a vertex object to store an entire vertex, or we can store each component of our vertices in a separate array, e.g. one array for positions, another array for normals, etc…
- We'll use the first approach here:

```
struct SimpleVertex{
  Vector3 position;    // position
  Vector3 normal;      // normal vector;
  Vector2 texCoords;   // texture coordinates
  Vector4 color        // color info
  };
```

Immediate Mode

- Immediate mode is the simplest way to pass vertex information to GL. It's what you've used in cs1566 so far
- It's fine for small applications in which performance is not an issue, but it's not efficient in terms of speed of memory usage

- Immediate mode syntax revolves around the glBegin and glEnd commands

- Inside of glBegin and glEnd blocks, we use the functions glVertex, glNormal, glColor, and glTexCoord to pass vertex information to GL.

```
// set render state and modelview, projection matrices

glBegin(GL_TRIANGLES); // tell GL we wish to begin drawing
    triangles
    for(int i = 0; i < 3; i++){
      // tell GL what color this vertex is
      glColor3f(colors[i][0], colors[i][1], colors[i][2]);
      // tell GL where the vertex is
      glVertex3f(positions[i][0], positions[i][1], positions[i]
    [2]);
    }
glEnd(); // tell GL we're done passing it vertices and are
    ready to draw
```

Retained Mode

- Immediate mode is simple to use but slow.

- Faster: use Vertex arrays
  - With vertex arrays, we essentially give OpenGL a pointer to a bunch of Vertex data and tell it to render from there.
  - We're no longer making function calls for each vertex
  - GL can take advantage of the fact that vertex data is laid out contiguously in memory
  - But there are still problems
    - We still have to transfer data from system memory to the GPU
    - What if we use the same vertex more than once? We have to store multiple copies of it in the array! What if memory is tight?

Even More Efficient Drawing

- Indexed Vertex Arrays
  - Now, rather than simply storing every vertex of every triangle in a long array, we only store one unique copy of each vertex in the array.
  - We then create an "index" array of indices into the first array that specifies which vertices to render.
  - The idea is that vertices take a lot more space than shorts or ints or whichever data type we use for indices.
  - If a lot of vertices are duplicated. We save a lot of space!

- Vertex Buffer Objects
  - OpenGL provides a mechanism for storing vertex data in fast video memory called a Vertex Buffer Object or a VBO.
  - Lets us avoid the cost of transferring vertex data from system memory to the GPU
  - Can be combined with Indexing for extremely fast vertex processing

- Display Lists
  - Allow us to precompile OpenGL commands for faster execution
  - Faster than VBOs ideally, but can't be used in all circumstances.
  - Data has to be static. Display lists can't be modified once they are created