Development Tools

CIS 487/587

Bruce R. Maxim
UM-Dearborn

Planning

- Games are not meant to have a 20 year life span so people are tempted to cheat on software planning
- Games have an artistic element which makes this type of planning more like the movie industry
- Risk management both proactive and reactive are at the heart of this planning

Reactive Risk Management

- Problems not addressed until they surface
- Why are problems overlooked?
  - Developer is overly familiar with the project and the product (day to day grind prevents seeing mounting problems)
  - Managers often try to shoot the messenger rather than deal with the message

Contingency Planning

- Most delayed games are victims of reactive risk planning
- Correctly implemented contingency planning adds very little work to project
- Goal of contingency planning is to save time and money
- Every project has problems the question is the effectiveness with which they are dealt with

Design & Architectural Problems

- Changes to base-lined requirements
- Poorly defined requirements often causes project scope extension
- Unfamiliar methodologies, tools, libraries
- Architectural integration problems

Schedule Threats

- Too tight schedules
- Incomplete schedules
- Unavailable resources
- Overestimation of schedule savings
- Inaccurate schedule estimates
- Schedule adjustment errors

Organizational Problems

- Unwilling to deal with subordinate criticism
- Management induced difficulties
  - Vested interests OEM products
  - Insisting on “right of veto”
  - Lack of disciplined development process

Contractor Problems

- Late delivery
  - Insist on penalties for late delivery
- Poor quality
  - Allow for time in schedule to review products and correct quality problems
  - Provide solid requirements statement to contractors
- Lack of skill or motivation

Personnel Problems

- Failure to work as a team
- Poor communications skills
- Low motivation or morale
- Skills shortages
  - Allow extra time to come to speed
  - Encourage knowledge sharing
  - Hire outside contractors

Development Problems

- Inadequate office facilities (not enough or not available when needed)
- Poor quality development tools or third party libraries
- Misinterpretation of design documents
- Failure to meet requirements
- You have to stop researching at some point in time and start developing product

Process Problems

- Excessive red tape and bureaucracy
- Failure to apply procedures correctly
- Failure to make use of data used to compute metrics in risk management
- Getting the level of formality to the right level (games are not life and death products)

State of Industry

- First era
  - Talented amateurs
  - Tie-in licenses introduced (e.g. Disney)
- Second era
  - Mass-market consoles
  - Many similar games sold
- Third era (current)
  - Rise of the 3D polygon market
  - Controversy heralds release of hot games

Celebrities

- Can be virtual (Laura Croft) or real
- Movies and merchandise follow release of several games and make more money than the game
- You can’t create likable characters and sidekicks by formula
- Recognizable likenesses must be distinguishable from other games

Violence

- Ratings are used to market games
- Players are disappointed when violence is not realistic
- Games do not cause violence in the real world (they don't prevent it either)
- In the long run games will need to do what television does and show the consequences of violent behavior

New Developer Model

- Leveraging intellectual properties
  - Games based on existing character licenses out sell “in-house” characters
- Developing several projects simultaneously is a better way to reuse technology than serial reuse of “old” technology

Commercial Alliances

- Technological resources and management can be shared to allow for economies of scale
- Umbrella of studio allows nutruing of newcomers
- Studio can maintain a talent pool inventory
- Super-developer can raise capital for development more easily enabling higher royalties for participants

Managing Outscourcing

- Creative core must maintain ownership through out project
- There must be standardization of design methods, libraries, and tools for this to work
- Current trend is for developers to pay development costs (publishers only want to pay for completed games)

Delivering Games Online

- The cost of the first level of a game might represent 50% of the total cost
- If the game were released on a pay for play basis over broad band the remaining levels could be released like chapters every few weeks
- The entire game could be released as a stand alone “greatest hits” as market is saturated

Playing Game On-line

- People would rather pay $5 for 2 hours of entertainment than pay $35 for 40 hours
- In 2 hour chunks a 40 hour game can net $50 to $100
- Massively multiplayer games have the potential to make a lot of money as well

Object-Oriented Design

- Programs written in procedural languages become impossible to maintain at 25,000 lines of code
- Programs written using object-oriented programming don't become hard to maintain until 100,000 lines of code
- Supporting the creation and reuse of subsystems is the main reason this is so

Pressure to Produce

- Games are sensory experiences so there is a lot of pressure to produce pictures and audio quickly
- This may make the development of coherent underlying architecture difficult and the prototype becomes the product
- As always the sooner you start coding the longer the development time once you start tracking maintenance time and costs

Object-Oriented Design

- OOD is successful because it allows developers to partition and package functionality more easily
- OOD allows developers to package components into reusable modules
- OOD has the promise of supporting the practice of component-based game development

Code Reuse

- It requires more time and resources to write truly reusable code
- There is a lot of informal code reuse in the game industry
  - DirectX (virtually every PC game)
  - Quake engines
  - Specialized engines (physics, 3D, AI)

Design Reuse - Patterns

- Design patterns are generic solutions to frequently encountered problems
- These solutions have worked well in the past and do not need to be reinvented each time they are needed
- Ultimately developers need ways to store and retrieve design patterns based on the problem attributes

Object Factory Pattern

- Allows the creation of a family of objects
- Each object has a serial number and is easily imported into new game levels

Singleton

- Ensures that only one object instance can exist in a game
- Probably instantiated during program initialization or on first use

Flyweight

- Allows user to instantiate multiple instances of a class that refer to a common shared component
- The only data stored in the flyweight object is the configuration information

Chain of Responsibility

- Sets up a chain of objects that receives notification of an event to avoid direct coupling
- Think of event handlers or call back functions

Iterator and Reverse Iterator

- Used to simplify process of lists of objects
- Often requires taking a pointer to a function as an argument
- C++ templates can occasionally provide this type of functionality

Template and Strategy

- The strategy pattern allows the dynamic selection of entire algorithms or book moves
- The template pattern is similar to the data driven techniques described earlier in the course

Observer

- Allows many to one object dependencies of the type found in blackboard systems
- One of each disadvantages is that the blackboard often needs to broadcast info to allow possible observers ‘just in case one of them is interested”

Command

- Allows the encapsulation of commands as objects so they may be passed to other objects
- This allows the creation and processing of command queues and menus
- This also allows the creation of scripting engines or interpreters

Decorator

- Formalization of a wrapper class that provides an interface to another class
- In general this is done to provide functionality not found in the wrapper class alone
- Might be similar to model-view-controller architectural pattern

Facade

- Provides a simplified interface to a set of related classes or subsystem
- Increases encapsulation and reduces code complexity
- This what LaMothe does with DirectX to simplify coding

Mediator

- Manages interactions among subsystems
- Façade communications are one way, mediator communications may be two way
- By encapsulating the interactions coupling it reduced
- Mediators are often implemented as observers

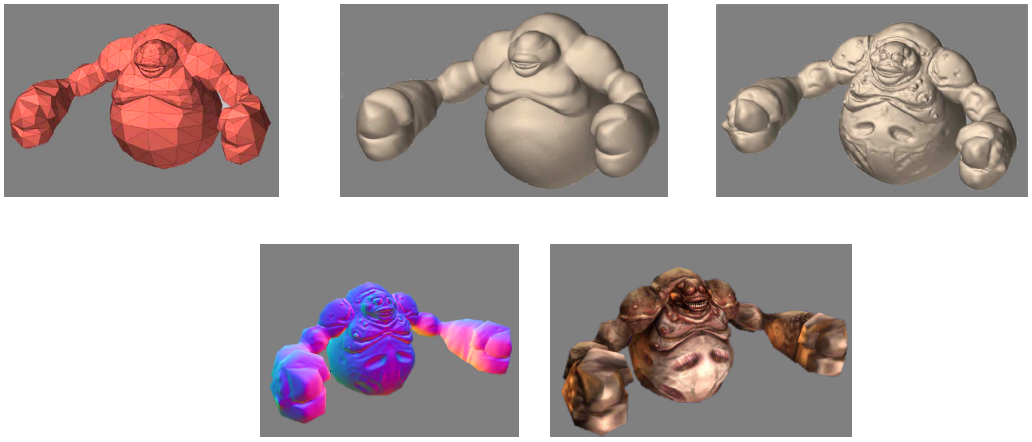
State

- Allows object to alter its behavior when its internal state changes
- Allows object to appear as new object
- This central to the ideas behind data driven game engines

The remaining slides come from the Rabin textbook

Other Modeling Techniques: 3D Sculpting

- A low resolution model can be sculpted into a very detailed mesh.
- This can be used in game via normal maps



Images courtesy of Pixolgic.

Other Modeling Techniques: Reverse Engineering

- Real world objects or sculptures can be scanned or digitized
- This may not save time because of complicated cleanup, but will ensure high fidelity

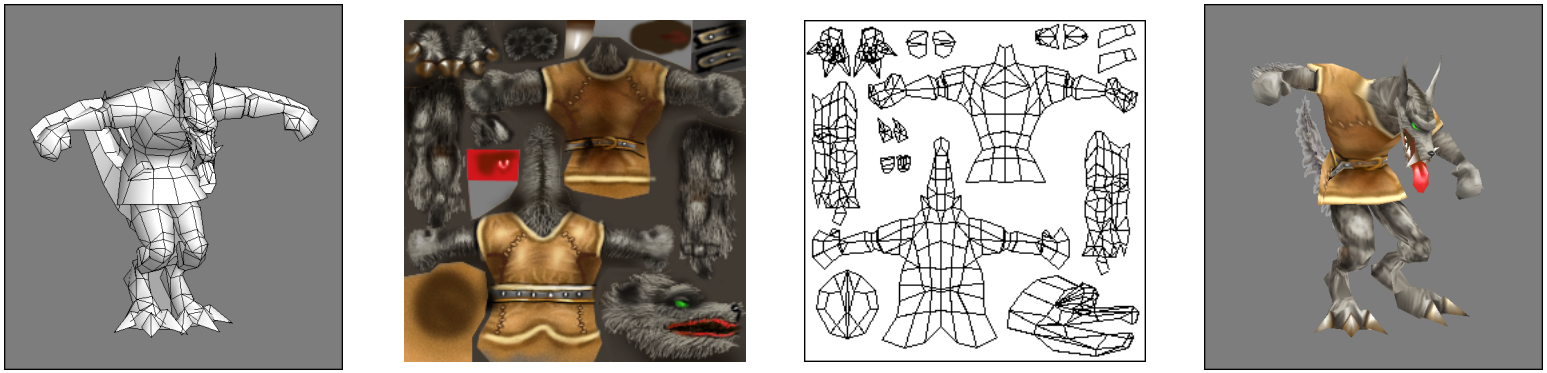
Image courtesy of FARO Technologies, Inc.

BSP Modeling Techniques

- BSP stands for Binary Space Partition.
- It's a coding term that is a method for organizing data
- BSP Editors come with many games like Quake, Unreal and Half-Life
- BSP modeling is like cutting away a mineshaft

Case Studies:
Low Poly Modeling

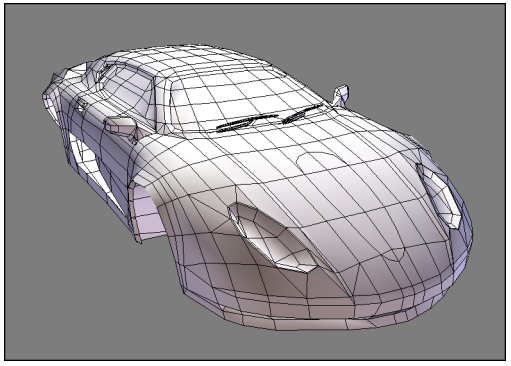
- With low polygon modeling, much of the detail is painted into the texture

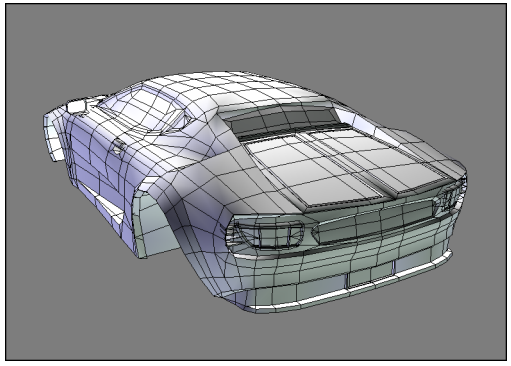


Images courtesy of WildTangent, model and texture by David Johnson.

Case Studies: Car Model

- Car reflections are extremely sensitive to topology problems and imperfect panel curvature
- Preview your model in the engine to troubleshoot reflection irregularities

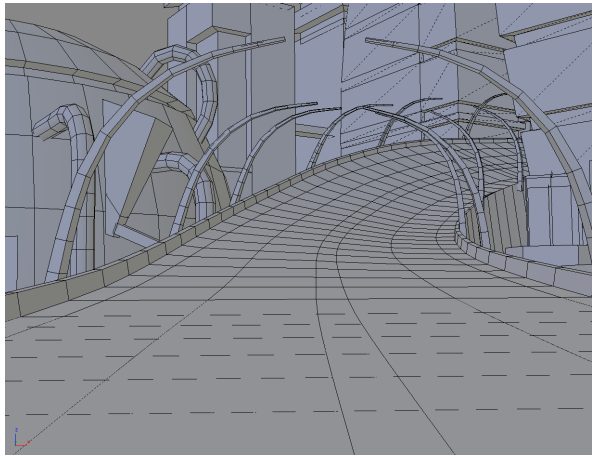


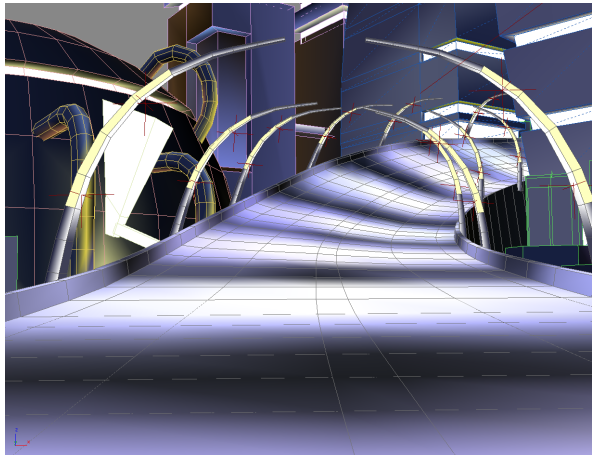


Images courtesy of WildTangent.

Case Studies: Environment Modeling

- An environment model showing stages:
  - Modeling
  - Vertex Lighting
  - In game with textures and effects





Images courtesy of WildTangent.

Volume Enhancements

- Flat screen requires exaggerating 3D aspects
- Enhance perceived volume in 3D with
  - Tunnels, tubes, spires, pits, towers, cliffs

Case Study:
Dark Orbit

- Uses tall rock spires and plunging canyons with fog to help accentuate volume in the 3D world
- Gameplay is restricted to 2D space (horizontal plane), but the volume of the environment is accentuated

Characters are confined to a 2D surface of play. Geometrical enhancements to the terrain are necessary to give the 3D effect.

Large rock spires sweep past the camera lens as the player moves throughout the world.

Image courtesy of WildTangent, Inc.

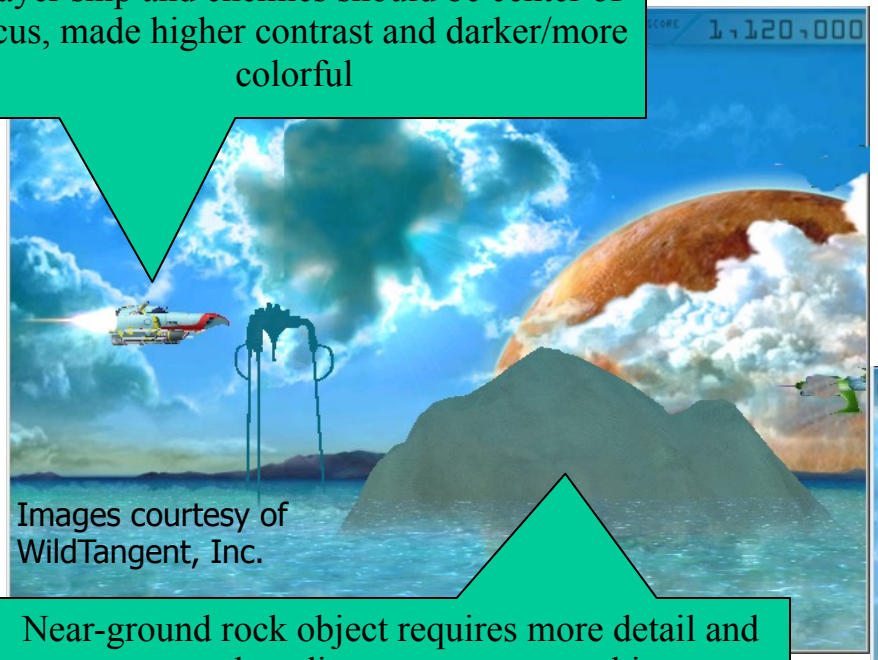
Distance Effects

- Near objects need more textural detail than far objects
- Distant objects should appear hazier and have lower contrast
- Warm colors perceived as being closer
- Cool colors perceived as being further

Case Study: Phoenix Assault

Before and after depth principles have been applied

Player ship and enemies should be center of focus, made higher contrast and darker/more colorful



Images courtesy of WildTangent, Inc.

Near-ground rock object requires more detail and contrast than distant water tower object

Distant objects should be made hazier and lower in contrast

Water should be made higher in contrast with reflections added to near-ground objects

Cloudy background has been made hazier and lower in contrast

Planet colors have been made cooler, haze has been added

Additional water tower has been brought closer to player and made darker

Faking Detail with Vertex Coloring

- Circumvent texture and poly limits with vertex coloring
- Popular and cheap technique
- Can give impression of
  - Shadows
  - Depressions
  - Variety

Case Study: Polar Bowler

- Uses a single, heavily-tiled texture on the ice walls in this level
- Vertex coloring is used extensively
  - Adds color variation and “fake” detail

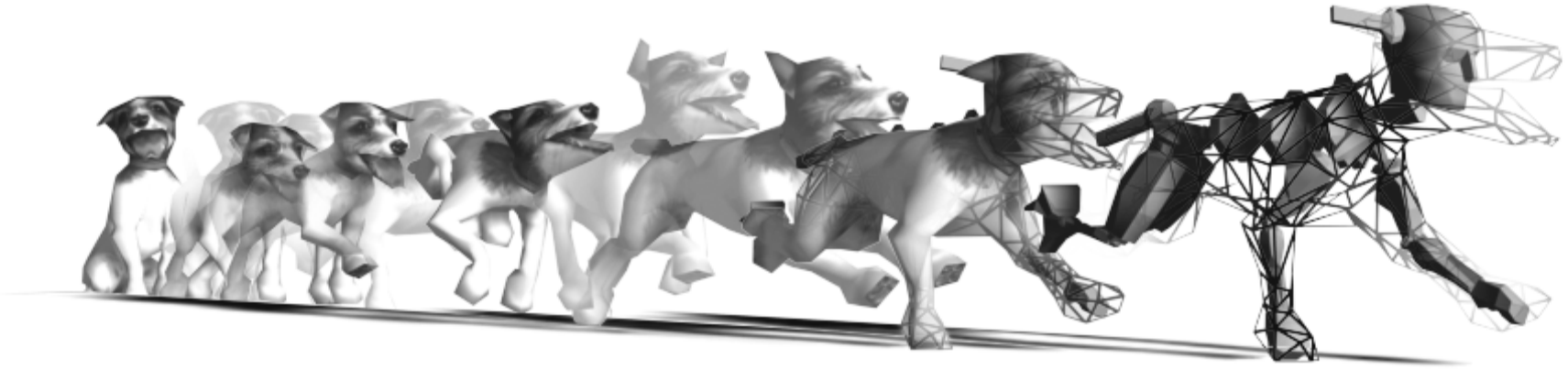
Warm vertex colors such as yellows and oranges give some warmth to an otherwise cold level, and also hint at the reflected light of the suns rays

Shadows created with vertex coloring can reduce the need for ‘real-time’ shading

Many crevices and undulations are implied by variations in vertex color banding along the geometry wall.

Image courtesy of WildTangent, Inc.

Character Animation



Overview

- When to use animation
  - Feedback to player about interaction with UI and in-game action
  - Communicating environmental conditions
  - Conveying emotion and expression in player characters and NPC
  - For visual appeal and dynamic interest

About the Animator

- Responsibilities
  - Hand animate simple and complex objects and characters
  - Set up character rigs (skeletons)
  - Export, test, and revise motions
  - Work with motion capture data
- Expectation
  - Understand animation principles
  - Design custom, often specialized moves that “read”

2D Versus 3D Animation

- Borrow from traditional 2D animation

Image courtesy of George T. Henion.

- Understand the limitations of what can be done for real-time games
- Designing 3D motions to be viewed from more than one camera angle
- Pace motion to match game genre

Production Workflow

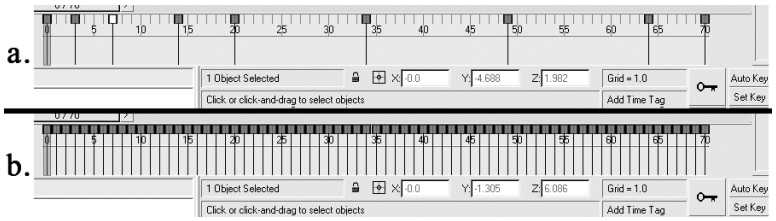
- Modeling and texturing objects to move
  - Mesh design to support bending
  - Single mesh vs. segmented mesh design
- Setting up a skeletal system
  - Proper placement of bones and joints
  - IK vs. FK systems
  - Setting up controls - pros and cons
- Binding the mesh to the skeleton

Binding a Mesh to a Rig

- Vertex weighting
  - How to assign and define bone influence
  - Adjusting envelops
- Testing and trouble shooting
  - How to revise vertex weight values to address problems
  - Finer single vertex control

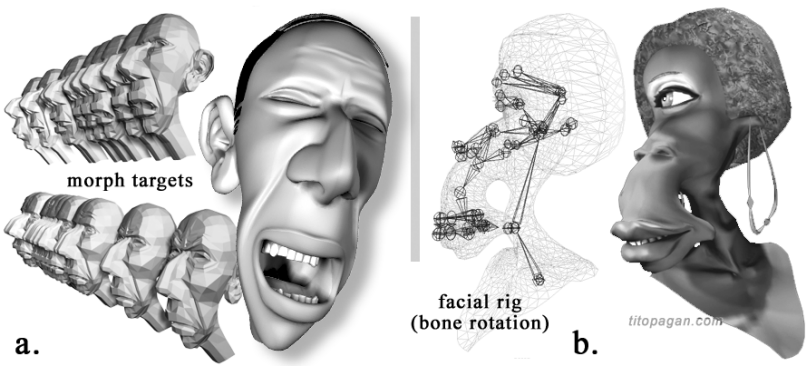
Keyframed Animation

- The timeline
  - Creating and adjusting keys
  - Adjusting playback speed
- Pose-to-pose approach
  - Extreme poses
  - In-between motion
- Kinematics systems
  - Forwards kinematics (FK)
  - Inverse kinematics (IK)



Facial Animation

- When to consider using
- Systems commonly used for setting up
  - Morph target set up
  - Skeletal rig system
- Trade offs of each
  - Realism
  - Controls
  - support



Left image courtesy of Nick Kondo, right image courtesy of Tito Pagan.

Motion Capture

- When to consider using *mocap*
  - Style considerations
  - Specialized moves
  - Cost effectiveness
- Technical understanding of data
- Preplanning for a “shoot”
  - Creating a well planned Move or Shot List
  - Knowing the pipeline to be used
  - Finding and directing skilled talent