CMP Game Group Presents:

Gamasutra.com

Technical Event Wrap Up - SIGGRAPH 2005

By Morgan McGuire
Gamasutra
August 11, 2005

URL: http://www.gamasutra.com/features/20050811/mcguire_01.shtml

SIGGRAPH is the most significant conference in computer graphics. It brings together game developers, film industry professionals, and scientists for a week in August. They present today's hottest products in each field and the new research techniques that will drive tomorrow's development. This article reviews the research papers presented this year that are the most relevant to game developers. SIGGRAPH also contains an exhibition, courses, technical sketches, animation festival, and special events. You can read about the non-research side of SIGGRAPH in Brad Kane's Event Wrap Up article here on Gamasutra.

Modeling

Most game companies write as much tool code as game code. That is because ultimately a game looks as good as its art assets, and better tools lead to better art. New advances in mesh and texture manipulation presented at SIGGRAPH regularly lead to improved in-house and 3rd party art tools years down the line. Read the modeling papers and get ahead of the curve—much of this technology is ready to be implemented now!

George Lucas gave the SIGGRAPH keynote. In 1977, *Star Wars* turned Hollywood on its ear and put sci-fi, fantasy, and special effects at the forefront when the industry as a whole was rejecting those ideas.

Explosions from Star Wars rendered by Selle et al.'s method, described in A Vortex Particle Method for Smoke, Fire, and Explosions.

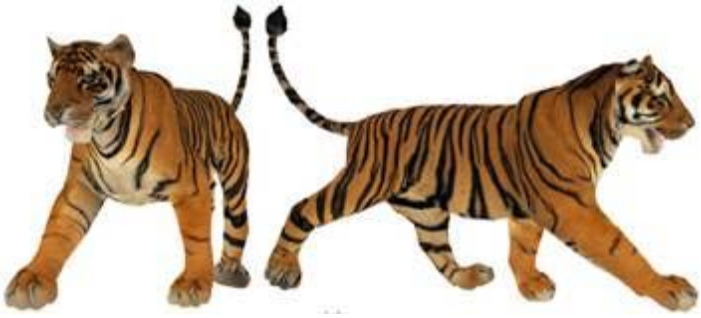
However, one of the more subtle ways that the original *Star Wars* film changed science fiction

forever was... dirt. *Star Wars* was the first film where spaceships and technology were covered in oil and grid instead of appearing as fresh, gleaming objects. Rust, stains, and dirt make objects appear part of their environment and suggest a history that began long before the observer arrived. Lucas points out that the more fantastic the story and setting, the more important it is to make it familiar and realistic so that the audience can relate.

This applies to games as well. Good 3D game artists create meshes and textures that are rusty and dinged-up like the spaceships in *Star Wars*. This year, Chen et al. introduced a new algorithm for helping with this process called **Visual Simulation of Weathering by Gamma-ton Tracing.** The “gamma-tons” are imaginary old-age particles that fall from the sky and bounce around a scene. Wherever they contact objects they induce aging artifacts like rust. Because the path of each gamma-ton is traced they are good for simulating effects like stains and moss-growth that flow outward from a source. The ideal use for gamma-tons in games is as part of a level compiler. After building a clean scene, the level compiler can crunch on not only light-maps and AI planning but on weathering the environment for realism.

Computed saliency of an action figure mesh (Lee et al.)

Game modeling is a constant tension between adding detail like dings and cracks, and removing polygon detail to make scenes animate and render faster. Lee et al. presented a paper that improves detail reduction. It is based on the idea of *saliency maps,* which describe the relative impact of different parts of an object to a human observer's perception. Previous salience algorithms have been directed at images. There, features like line intersections, text, and human faces are marked as important (salient). Lee et al.'s **Mesh Saliency** advances the state of the art by describing the first algorithm for computing saliency on a 3D mesh. This is exciting news for artists. The result opens the possibility of automatic LOD algorithms that reduce polygon count while maintaining the perceived shape of an object. In the paper, the authors give examples of automated mesh simplifications that appear significantly better than results produced with conventional algorithms. Programmers should be able to directly implement the paper's methods for in-house use (particularly for normal-map generators) and hopefully salience-based simplification will soon appear in modeling tools.



Tiger mesh textured from photographs by Zhou et al.

Zhou et al.'s **TextureMontage:Seamless Texturing of Arbitrary Surfaces From Multiple Images** describes a tool for extracting textures from photographs and mapping them seamlessly onto 3D meshes. They demonstrate texturing a tiger from two pictures using only a few brush strokes and then repeat the process for a kitten and a zebra. Occasionally more interaction is required to patch holes along a model's underbelly or inside complex features. Yet the method appears far superior to manually editing (u, v) values in Maya or 3DS Max and generates great results for the models shown.

Rendering

Fabio Pellacini's paper on **User-Configurable Automatic Shader Simplification** describes a method for simplifying pixel shader code while preserving high-level effects. Simple shaders are useful for efficiently rendering far-away objects and for automatically reducing detail on low-end computers. The system analyzes mathematical expressions in the shader and replaces them according to a series of rules. The rules contain simplifications like “the sum (a + b) can be replaced with (a).” The average value of any expression (like a noise function) can also be substituted for that expression. The system is demonstrated producing automatic simplifications for some straightforward procedural textures—more testing will reveal whether it can also simplify the complex shaders found in today's games.

Ken Perlin's noise generator has long been a staple algorithm in computer graphics. It is used to create semi-random textures like clouds, lava, and dirt and appears in almost every game engine and modeling package. In **Wavelet Noise** presents a new noise generator by two well-known graphics researchers from Pixar, Cook and DeRose. The wavelet noise preserves the good properties of Perlin noise but adds improved high frequencies without aliasing (swimming) artifacts. The new algorithm has about the same run-time cost but requires about 250 times as much memory. The appendix gives C source code for implementing the new noise generator. This is helpful, given that the rest of the paper is loaded with complex integrals and math notation.

Precomputed radiance transfer has been popular in games and research for the past few years. The basic idea of the approach is to pre-compute the form factors that describe how surfaces in a scene interreflect light, but stop short of computing a final light map. When lights move in the scene realistic indirect lighting can then be rendered in real-time. The appeal of the basic technique is great real-time lighting; the drawbacks are tremendous memory requirements and that the technique is ineffective for scenes with moving objects. Several papers this year propose new additions to PRT to allow dynamic scenes and new lighting effects.

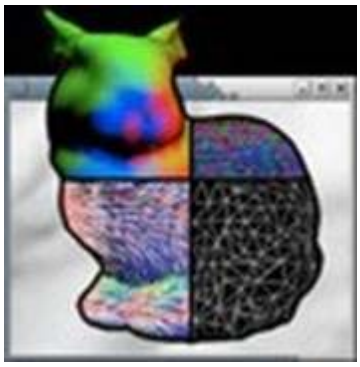
Precomputed shadow fields by Zhou et al.

Zhou et al.'s **Precomputed Shadow Fields for Dynamic Scenes** precomputes the shadowing effect of an object on its environment. The objects can then move (but not animate) relative to each other and have. The technique requires hundreds of megabytes of memory and renders slightly slower than real-time, but the ideas behind it are interesting and it is possible that future work could make something like this practical for games.

Wang et al.'s **All-frequency Interactive Relighting of Translucent Objects with Single and Multiple Scattering** adds precomputed sub-surface scattering terms that allow almost real-time rendering of subtle lighting effects from skin and marble. Kristensen et al.'s **Precomputed Local Radiance Transfer for Real-Time Lighting Design** supports with moving local lights by determining the most significant lighting components. By reducing lighting to as few as eight components per vertex, they achieve real-time rendering.

Most significant for games, Sloan et al.'s **Local, Deformable Precomputed Radiance Transfer** is the first PRT approach to support deformable models in real-time. They use a new "zonal harmonic" scheme for encoding the transfer function and show how to deform precomputed lighting to allow objects to animate. In one demo the authors show a fully animated bat rendered at 240 fps with sub-surface scattering, soft shadows, and interreflection! The approach only works for coarse, diffuse illumination and is only good for extremely local effects. For example, a character's forearm will shadow its elbow, but not cast a shadow on the ground.

The PRT challenge for next year is an algorithm that combines the non-local shadowing of the shadow field with the local, deformable lighting for animated characters and compresses it into tens of megabytes, not hundreds. Until such an algorithm arrives, PRT remains impractical for widespread use in games but is an area to keep an eye on.



Four debugging visualizations produced by Duca et al.

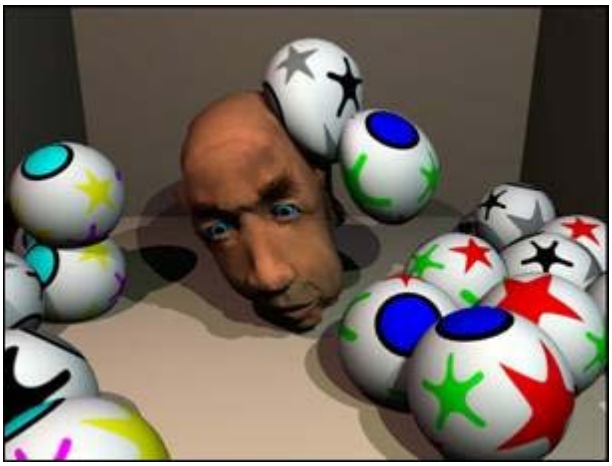
Graphics hardware makes games run fast—but it makes developing them very slow. The problem is that traditional software tools like debuggers and profilers only run on the CPU and can't help with coding running on the graphics processing unit (GPU). This year Duca et al. introduced the first real debugger for GPUs. **A Relational Debugging Engine for the Graphics Pipeline** intercepts all OpenGL calls and sends them both to the debugger and to the GPU. The GPU renders the scene normally while the debugger builds a relational database of scene information. To debug, the programmer issues SQL-like database queries. The debugger displays the results of those queries as wireframes, color-coded renderings, textures, and other visualizations. This allows programmers to debug graphics applications at runtime and without modification. It is the difference between inserting "printf" calls in traditional code and using a real on-line visual debugger.

Physics and Animation

Modeling and rendering produce great screenshots. But a game is not its screenshots—physics and animation bring images to life and make games compelling. Real-time physics was unheard of a few years ago, and is now well represented through middleware and game engines. Almost every new title ships with some realistic physics and many incorporate physics directly into the puzzles and character animation that are the fabric of the game. This year's physics and animations papers address both speed and quality concerns, making physics more realistic and lifelike without sacrificing performance.

Kaufman et al.'s **Fast Frictional Dynamics for Rigid Bodies** simulates non-linear friction on rigid bodies extremely fast. They demonstrate correct shock propagation, rolling, sliding, stacking, and tumbling, which are areas that physics solvers traditionally perform poorly in.

The secret to their approach is a single model that unifies rolling and sliding friction, and they get good performance by considering all contacts simultaneously. Their results are not real-time, however they consider scenes containing thousands of objects in contact and implemented their simulator in Java. It is likely that an efficient C++ implementation could run in real-time for the smaller scenes handled by games.



Muller et al.'s deformable models take a beating in real-time.

Muller et al.'s **Meshless Deformations Based on Shape Matching** had the best game-related demo. The authors showed a real-time simulation of a head buffeted by toy balls where all surfaces were deformable and rendered with multiple light sources. The demo was completely stable and combined the quality we associate with offline simulation with the interaction of a game. It provided a completely convincing virtual world that begged to be played in. Look at the videos on their website—if you're a physics programmer you'll want to stay up late tonight to implement the algorithm! It is easy to code and gives great results. The algorithm computes deformable physics using a relative of the familiar mass-spring model. Selected vertices are assigned canonical positions. When a vertex is deformed from its canonical position, that both applies a force on the whole body and introduces a spring force that attempts to pull the vertex back into position. The entire algorithm can be expressed as a fast linear system inside a traditional physics pipeline, making it possible to handle upwards of 10,000 points in real-time.

Physics skeleton by Redon et al.

For fast physics on characters with joints, read Redon et al.'s **Adaptive Dynamics of Articulated Bodies**. Their algorithm examines characters and identifies joints that can (temporarily) be approximated as rigid without altering the overall motion significantly. This allows simulation that is 10 to 100 times faster than previous approaches. Without motion capture or keyframe driving them, those characters will just be rag-dolls. Zordan et al.'s

Dynamic Response for Motion Capture Animation combines a contact force physics simulator with motion capture data. This allows a smooth and controlled transition from a pre-animated motion like a defensive karate block to the simulated fall when the block fails to stop an incoming attack. Unlike some papers, Zordan et al. are very clear about their applications. The characters in their animations literally perform karate moves and demonstrate a variety of life-like reactions including falls, tumbles, and even push-back from their own attacks connecting. This technique can be the difference between seeing a stock damage animation and ultra-realistic specific damage and tumbling that accurately reflects the environment. They showed many cases where the same attack led to visibly different damage animations, in some cases with the defender rolling out of the hit and recovering. Unfortunately, this algorithm isn't ready for prime-time in games just yet. It is very fast but occasionally produces hiccups, possibly because it depends on linear blending and doesn't match momentum of all body parts. Look for follow-up work soon that addresses these problems, because the initial results are too good to let this fall by the wayside.

Conclusions

Preprints of many of this year's papers are available on the web collected at http://www.cs.brown.edu/~tor/sig2005.html, and all SIGGRAPH content is available via registration from the ACM Digital Library at acm.org.

Next year SIGGRAPH moves from LA to Boston . There are many games industry challenges that would be appropriate to see solved there. Just as shadows went from an exotic to must-have feature in the last five years, physics now looks like it will be standard in upcoming games. Previous physics papers solve a problem beautifully but slowly (e.g., the crop of water and smoke papers that take hours per frame) or solve it well but for a limited domain, like the articulated and deformable characters discussed in this article. Is it possible to perform the good (even if approximate) water simulation in real-time? How about a character that is deformable *and* articulated *and* animated *and* fast at the same time?

Fast global illumination remains a huge challenge. We need plausible and blazingly fast illumination to make environments feel real in games, and Kristensen et al.'s real-time lighting is a good start. Next year let's hope for the next step, with higher performance and less precomputation.