GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: DreamWorks Interactive's *Trespasser*

By Richard Wyckoff

Editor's note: This Postmortem appears in the June issue of* Game Developer *magazine. Due to space restrictions in the magazine, we were forced to shorten it somewhat. This article contains quite a bit of additional information not printed in the magazine version.

"One seldom hears the true story of what happened at the place where the world changed. How it began. What were the reasons? What were the costs?" -John Parker Hammond

This quote from *Trespasser*'s intro movie serves just as well to open the real story of a game development team's struggles to develop a breakthrough dinosaur game as it does to open the fictional story of Hammond's struggle to develop a biotechnological breakthrough and clone dinosaurs. The parallels between the *Trespasser* project and Hammond's cloning project were numerous: ambitious beginnings, years of arduous labor, and the eventual tragic ending. Hammond's diary, as related in the game itself, dwells on the past and never attempts to explain Hammond's future direction now that he has failed so grandly - this postmortem is intended to be much more forward-looking.

Trespasser was begun by two former employees of Looking Glass Technologies, Seamus Blackley and Austin Grossman. By the time the game was rolling, two more ex-Looking Glass employees would join the team, and our common background was instrumental in setting the direction for the project. Looking Glass's most distinguished products, *Underworld I* and *II* and *System Shock*, are games which in some ways are still ahead of their time, specifically in the areas of object-rich, physics-based environments and emergent gameplay.

Quake did not even ship until after coding on*Trespasser* had begun, and to the *Trespasser* team with its founding in Looking Glass's design-focused philosophy, it represented the stagnation of 3D games rather than the step forward it was proclaimed in the press. *Quake* did nothing to extend the basic first-person shooter game design standards of "find weapons and keys" which id had first created in *Wolfenstein 3D*, and replaced the fairly-consistent atmospheres of *Wolfenstein 3D* and *Doom* with a bizarre mishmash of medieval and science-fiction themes. *Trespasser* was intended to be a high-technology game where game design and world consistency came first.

***Trespasser* concept sketch**
[zoom]

The Jurassic Park license was inevitable from the start, for a couple reasons. The obvious reason was that *Lost World* was on its way and expected to be a gigantic hit, and standard Hollywood thinking dictates that all projected hits be exploited seven ways to Sunday. The less obvious reason was that Seamus had been working on a physically-simulated biped model originally intended for Terra Nova, and had been shopping it around to several movie animation groups working on dinosaurs before ending up at DreamWorks Interactive.

The pie-in-the-sky concept for *Trespasser* was an outdoor engine with no levels, a complete rigid-body physics simulation, and behaviorally-simulated and physics-modeled dinosaurs. The underlying design goal was to achieve a realistic feel through consistency of looks and behavior. Having an abandoned island setting was a useful way to exclude anything which did not seem possible to simulate, such as flexible solids like cloth and rope, wheeled vehicles, and the effects of burning, cutting, and digging.

The game would play from a first-person perspective, and you would experience the environment through a virtual body to avoid the "floating gun" feeling prevalent in the *Wolfenstein* breed of first person games. Combat would be less important than in a shooter, and dinosaurs would be much more dangerous than traditional first-person shooter enemies. The point of the game would be exploration and puzzle-solving, and when combat happened, it would more often involve frightening opponents away by inflicting pain than the merciless slaughter of every moving creature.

"Limited but rich" was a phrase which was used often early in Trespasser's development. This phrase describes a game design philosophy consisting of choosing a reduced feature set, but putting more sophistication into each feature. Although solid-body physics based entirely on

box-shaped solids might seem like only a rough approximation of the real world, the thinking was that a perfect simulation of solid boxes would be so much more flexible than the emulated physics of previous games that our gameplay would be deep and absorbing.

***Trespasser* concept sketch**
[zoom]

Likewise, though we would only have a few different types of dinosaurs, the dinosaur AI system would allow them to react to each other and the player in a large variety of ways, choosing appropriate responses depending on their emotional state. Sophisticated, fully-interruptable scenes would occur spontaneously rather than requiring large amounts of scripting, and observing the food chain in action would be as absorbing as playing the game itself. Interacting with the limited but rich features would lead to "emergent gameplay," the grail for many of Looking Glass' best thinkers since *Underworld I* shipped and fans began to write in describing favorite moments - moments which had not been specifically designed or even experienced by the team itself.

The original plan for *Trespasser* certainly seemed like a good one. It was very ambitious, but the team had made tradeoffs for implementation and execution time from the very beginning, such as not attempting to do multiple or moving light sources or *Quake*-style shadow generation in order to accommodate arbitrary numbers of moving objects and long, wide-open views. Unfortunately, there is a difference between having a plan and successfully executing it, and the product that we eventually shipped was as disappointing to us as it was to the great majority of game players and game critics.

Things that went right

Some have dismissed *Trespasser* altogether because it was such a visible failure. Respected columnists and editors use it as a reason why physics is bad, or make it the butt of their jokes ("at least it wasn't as bad as *Trespasser*!"). However, from a project perspective there were a number of successes. Before we get into the problems which ended up sinking the ship, let's look at these successes.

1. Use of license

From my own perspective as a designer, licensed properties are the least fun games to work on. Even Star Wars, perhaps the only movie license just about any game maker might donate a limb to be allowed to use, brings with it a huge amount of restrictions which can make that initially attractive license turn into a lead weight over the course of development. From a game player's perspective, the vast majority of games with movie licenses have been just plain awful. And finally, from a movie aesthete's view, the Jurassic Park movies are among Spielberg's (and original novelist Crichton's) worst work, lacking in just about anything beyond pacing and special effects.

However, *Trespasser*'s Hammond diary actually contains lots of interesting tidbits about the early days of characters like Henry Wu (the scientist from the beginning of the first movie) and even Dennis Nedry (Wayne Knight's character who basically caused the first disaster). The player can also check out locations on the island which imply back story which isn't explicitly told, like Henry Wu's house with its 80's executive bachelor stylings, Nedry's office with its poster for the fictional computer game series "Swords of Kandar," and Hammond's lavish mansion. The settings and the diary itself serve to reveal much of Hammond's motivation and personal reactions to the building of Jurassic Park, creating more of a character than exists in either the books or the movies (Crichton killed Hammond off in the first book, anyway).

The overall plot of our game is as simplistic as most others (Character finds way to escape death trap), but the details revealed about the *Jurassic Park* world extend it in a way which is faithful to the originals. Although it is quite likely that the next *Jurassic Park* movie to be released will make *Trespasser* non-canon, for now it stands as the only real extension of the series published. If there are such things as *Jurassic Park* fanatics and they were able to look past the gameplay flaws, they hopefully enjoyed our development of the world.

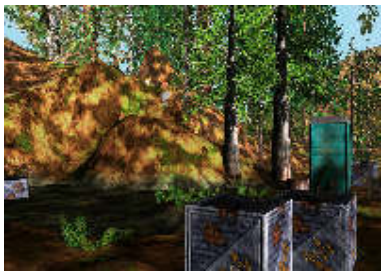
2. Art and music

On an individual basis the models created for *Trespasser* rank with the best-looking work done for computer games. We limited the largest texture size to 256x256 pixels, and at model import time textures were converted to 8-bit paletted images, but artists worked with their models using 24-bit art in 3D Studio Max, applying the textures using any mapping methods that Max supported. We had the standard limits on visible polygons, so most models were made with as few polygons as possible, with dinosaurs ranging from 300-500 and trees from 50-120 for example, but this still gave our artists more complexity than was standard at the time.

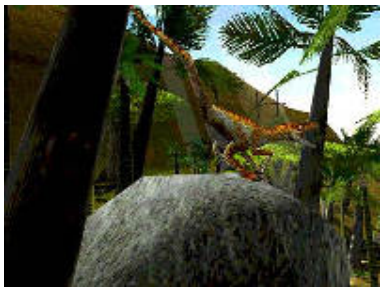
Many of *Trespasser*'s artists had never worked on games or done 3D modeling before, and some had never even used computers at all. This was a fairly deliberate decision, in an attempt to achieve a much higher standard of art than we were used to seeing on previous products. The number and resolution of textures we were able to support called for painting skills far beyond the average game-trained artist.

The music is one of *Trespasser*'s best accomplishments. Trespasser was not able to use the Jurassic Park license for free, despite Spielberg being a principal of our parent company, DreamWorks. A large chunk of our budget went just for that license, yet that money bought little more than the ability to use the name, locale and set designs from *Lost World*. Everything else that a reasonable person would assume would be part of a licensing deal, such as the amazing ILM dinosaur sound effects and the John Williams theme, would cost extra. Luckily, our sound effects were being done by professional effects company SounDelux, and they put us in touch with one of their stable of composers who specialized in "imitation" music. With very little prompting, he recorded about 30 minutes of music for us which in some parts far exceeds the rather forgettable work Williams himself did for the *Jurassic Park* movies.

Some reviews still accused us of having spotty or inappropriate music, but this was more an implementation problem than a problem with the music itself. The music was recorded as a couple dozen short sections which were scattered through the world on location-based triggers. Much like the voiceovers, more attention could have been paid to their placement so that they only played at appropriate and regular intervals. Even more desirable would have been a system with the ability to play tension or combat music loops and fade them in and out of the special-event songs to make it seem more like a continual musical score.



***Trespasser* screen shot**
[zoom]



Velociraptor read to pounce
[zoom]

3. Innovative systems

Our artists were able to paint textures with near-total disregard to common memory-conservation practices thanks to the texture caching system which was created fairly late in the project by one of the last programmers to be hired.

Textures for our game were MIP-mapped by our helper application as part of the process of building level data (curved bump maps were also created at this time). A level could have a nearly-unlimited amount of textures, and once MIP levels were created, all textures and their MIPs were saved into a single swap file. The app automatically organized the swap files into pages based on texture size, with the lowest couple of MIP levels for all textures on a set of pages which were always committed.

As the player moved through the level and objects came into view, the appropriate pages from the swap file would be accessed. In any circumstances where an appropriate texture hadn't been loaded in yet, the always-committed MIPs could be used until the higher-resolution texture had been loaded. In theory, this could result in a frame or two where an object was textured at a lower resolution than desired, but in practice it rarely happened, even on the most texture-intensive levels.

Another major system for *Trespasser* was its audio system, which we described as "real time Foley" because of its ability to generate collision and scraping effects between differing sound materials in real time. Although the system could have used more sound material data, even with what it had it resulted in some wonderfully immersive sound effects which most other games do not duplicate - things like scraping a board down a concrete surface or hitting an oil barrel with a metal bat sound just about perfect. Since the system doesn't simply play two sound effects but actually chooses from a group of samples and sets volumes based on the underlying physics collision, it sounds much more natural than most other audio systems currently used.

Finally, our image caching system, which rendered groups of distant objects into single 2D bitmaps to speed rendering, while responsible for the most-disturbing visual anomalies in the game, was also by its own right an amazing piece of work. Image caching made it possible to render scenes with tens of thousands of polygons in near-real time, and it is the first technology which allowed outdoor scenes to have a reasonable fraction of the complexity of the real outdoors.

4. Outdoor level design

When we created our terrain geometry, we were deliberately trying to avoid the "marbles in rubber" look of a lot of bad fractally-generated outdoors terrain. To this end, we decided that we needed to base our island on real-world terrain rather than build it from scratch. Luckily, we had a real-world model to go from: Costa Rica's Isla Sorna, the same island Crichton used as his inspiration for Site B. Unfortunately, no relief maps of sufficient detail existed for Isla Sorna, so we ended up having our lead artist sculpt a large model of the island, had it laser scanned, and did all our work on it in 3D Studio Max.

Outdoor levels are a hallmark of *Trespasser*

[zoom]

The hope was that the scanned model would provide enough fine detail of the island that we could approach it as if we were InGen's engineers and simply "bulldoze" a few roads and building sites into it. As it turned out the scanned model was only useful to set the rough contours for each area, and gameplay-relevant detail such as sight- and movement-blocking hills was largely hand-built. However, having an idea of where all the ridges in the island lay gave clear direction in the level-design sessions for how the gameplay would flow and what would happen in each area. The scanned model also provided continuously rolling terrain rather than a flat base with mountains sticking out of it (as hand-built outdoors areas tend to resemble). The number of slopes did increase difficulty of modeling the gameplay areas, but it was a worthwhile tradeoff for a natural feel.

Modeling was only the first step on a level: after the terrain was mostly established, there was a significant time investment in populating the levels with objects. A typical level consisted of 10 - 15,000 trees, shrubs and rocks, and a few thousand man-made objects as well. Every object could be placed individually, but for time-saving reasons we generally used groups of objects for areas off the game path and only spent a lot of time hand-placing items in places we knew the player wouldfrequent.. The rolling terrain and a random-delete tool we often ran for optimization generally kept the repetition from seeming obvious.

In the end, though our levels didn't quite fulfill our personal expectations, they usually look more like real environments than previous games. Starting from real terrain seemed to be the most useful tactic for this, and creating a natural algorithmfor vegetation placement is the next obvious step for outdoor games.

5. Realistic physics in first person

The first discovery we made as our physics simulation was slowly implemented was that it was an engrossing toy. When we finally got support for compound object physics (so that a bench could consist of a top and two legs instead of a single block, for instance), it was possible to spend an hour just dropping the bench onto things to see how it would catch on edges and flip and slide around. Toys do not make games, however, and trying to establish a gameplay which would work with the simulation was our major challenge.

Due to the vagaries of our particular physics simulation and interface, we eventually arrived on gameplay that primarily involved knocking things over rather than stacking things up. Knocking over will probably be the first application of realistic physics to see wide-spread use, as it will work even with a less-than perfect physics model such as *Trespasser*'s. It is also a behavior which easily shows off the difference between realistic physics and what we usually referred to as "fake" physics - compare pushing a box off a ledge or hill in any game which actually lets you move boxes to the same action in *Trespasser*.

Since our gameplay was supposed to revolve around the physics, however, we needed to apply that knocking-over behavior in slightly more sophisticated ways than just using it as eye candy. The best uses that we found in the small amount of time we had involved knocking stacks of boxes down to fill holes, or unbalancing something resting on a high ledge in order to get it or use it as a step. It also quickly became apparent that building even a simple-seeming knocking-down puzzle required a much more highly refined sense of physical laws than is common, and many designers ended up with a collection of blocks and small boxes to aid in brainstorming puzzles.

We had originally intended to have much more interesting physical challenges for the player consisting of making stacks and bridges to cross gaps or reach high players. Due to the problems with the physics engine, we ended up cutting nearly every puzzle which depended on this behavior, as it was too difficult and unreliable to actually make stacks. However, as it seems like such an obvious thing to do (and was so central to the design at one point), many players still try to make stacks, and come away from the game extremely frustrated. In different circumstances this constructive gameplay should be even more interesting than the purely destructive gameplay of knocking things over, and making it work ought to be one of the foremost goals of post-*Trespasser* physics engines.

Things that went wrong

It might seem like *Trespasser* deserves more entries in its "what went wrong" section than the usual project postmortem. However, *Trespasser*'s failings are actually numerically few. Unfortunately, the failings that we did have were serious enough to more than outweigh the successes we made.

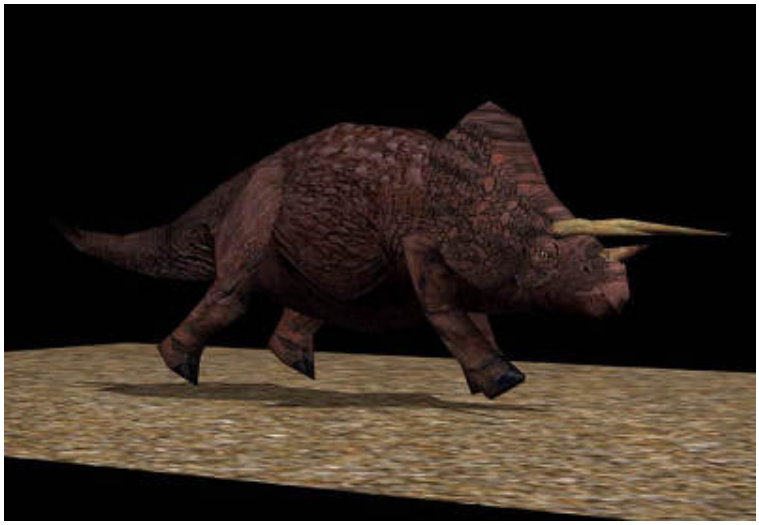
1. Software-oriented renderer

Trespasser was begun before the 3dfx Voodoo 1 started the wave of 3D hardware popularity. As such, the *Trespasser* engineers set about to create an engine in the old-school manner: they picked some previously-unseen rendering technologies and implemented them, ignoring any issues of compatibility with hardware cards. Our engine's two most incompatible features were its bump mapping, a true geometrical algorithm which could take surface curvature into account, and its image caching. Image caching was intended to allow real-time rendering of huge numbers of meshes, but it was also the system almost solely responsible for the graphical anomalies of popping, snapping trees. The other primary visual artifact of the game was the frequent sorting errors, but this was a result of poorly-constructed levels which continually handed our depth sort algorithm more polygons than its limit, and not a direct result of image caching itself.

The image cache system worked by rendering distant objects into 2D bitmaps on the fly, updating them when the angle or distance changed enough that the 2D representation was no longer sufficiently accurate. This may sound familiar to those who remember Microsoft's Talisman architecture, and there was a hope at one time that our game would be a killer app for Talisman accelerators, but resistance from key figures in the industry and 3dfx's sudden popularity pretty much put the end to Talisman.

Trespasser ended up slipping by more than a year, as did many games of its time. Our hardware programmer put in a valiant effort in the last half year of the project, and managed to get much more use out of 3D hardware than we initially thought possible. We ended up with a fairly unique mixed-mode renderer which drew any bump-mapped objects in software and the rest of the scene with hardware. Unfortunately, the large number of bump-mapped objects present in any scene of our game, such as all the dinosaurs and nearly every crate, meant that the fill rate advantages of accelerators were often negated.

Even in hardware-only scenes, PCI cards like the popular Voodoo2 became easily bogged down by the sheer amount of textures *Trespasser* uploaded as image caches were updated and the texture cache system loaded in higher mip-levels. This was simply too much, and acceptable hardware performance was often only possible by using textures that were less-detailed than in the software version, a unique situation but not surprising given the software origins of the engine.



***Trespasser's* software-oriented renderer allowed for detailed models [zoom]**

In addition to being a costly software-only rendering method, our bump mapping was never very evident: we could have used multiple, moving light sources and better use of the bump maps by the art staff. Many of the bump maps were created by simply converting the original texture to grayscale, an artist's hack that works for rendered images and animations but not in real-time 3D. Image caching was an even bigger problem than bump mapping, because though it was a key technology to allow the scene complexity we were attempting it was also the source of most people's visual complaints about the game. It seems clear in retrospect that we needed to make a tradeoff somewhere along the line and either drop the physics technology and physics gameplay in favor of the rendering technology, or more likely drop the rendering technology and ability to do complex scenes in favor of the physics technology.

2. Game design problems

The biggest indication that *Trespasser* had game design problems was the fact that it never had a proper design spec. For a long time, the only documents which described the gameplay were a prose-based walkthrough of what the main character would do as she went through the game, and a short design proposal listing the keys which would be used and some rough ideas of what gameplay might actually be. These documents were created before any playable technology existed and were based on promises of how that technology was supposed to work.

When the game had a complete team and had essentially entered production, the prose walkthrough was used to create level maps and more-complete puzzle descriptions. However, by this point in the project artists had been building assets for nearly a year and programmers had been implementing code for even longer, and the gameplay was being crafted as a primarily engineering-driven rather than design-driven process. Engineering-driven software design can work fine when the gameplay needs of the final product are extremely light and flexible, but in the case of Trespasser, where we were trying to achieve a specific and complex gameplay, it ended up being the source of many of our problems.

It is true that a design specification for a 3D game is rarely as complete and accurate a description of the final product as a spec for a 2D adventure or similar well-understood type of project. Nonetheless, our experiences on *Trespasser* made it clear that it is worse to not have a design spec at all than to have one which becomes out of date and is frequently rewritten. Design should really set the direction for all other development of a game, as no amount of programming or art can suffice for a lack of gameplay, but *Trespasser* started and finished weak in the game design, and this affected every other part of the project.

When it became clear that the technology was not going to exist to support the initial high-concept design, it would have been best to throw out all our existing notions and reinvent the game. Unfortunately, our license made it all but impossible to throw out the original *Trespasser* concepts. The only major deviations from the original concept were the change from constructive, stacking-based physics puzzles to destructive, knocking-over puzzles, and an attempt to make combat more prevalent in order to shore up the weakness of the destructive physics puzzles. Since no part of the *Trespasser* code was written to be good at doing first-person shooter gameplay, this attempt to make shooting a more important feature only ended up flaunting some of the weaker points in the game like the lack of an inventory system and the slow frame rate.

3. Tools Problems

Trespasser was built entirely in 3D Studio Max. There was no level editor, only the generically-titled GUIApp, which was the game with a debugging shell and not really a tool at all. Our level creation procedure consisted of arranging 20,000 - 25,000 meshes in Max. We used dummy meshes to represent gameplay objects like triggers and defined game behavior of all objects by typing code into their object properties buffers.

There were two unexpected but incredibly severe drawbacks that we discovered only after it was far too late to change our method: the first was that Max is basically unfit to work with more than about 5,000 objects at a time. The Max files for a *Trespasser* level averaged 40mb in size, and could take a couple minutes to load even on the 256mb PII-266s the designers used. When all objects in a level were visible, it could take 30-60 seconds to respond after clicking on an object to select it, making fast work difficult to say the least.

The second problem with the Max method was the use of the object properties text buffer. The buffer seems to be one of those features which no one ever used before, because we discovered that if more than 512 characters were typed into an object's properties buffer, Max could become unstable. If Max didn't crash outright and a file was saved with one of these bad objects, it would become unloadable. *Trespasser*'s technical artist wrote many MaxScripts which served as design tools and warning scripts to guard against problems with the properties buffer, but as they could only work passively and not actively, they made designing in Max tolerable but not enjoyable.

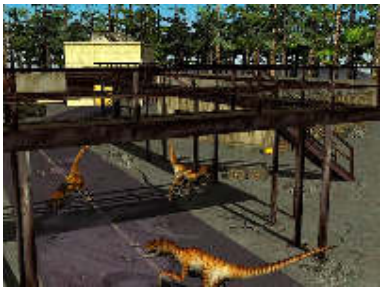
There was an additional wrinkle to the process of using Max to create levels, and this was the exporter step. A Max plug-in converted data into the game format, but our particular exporter caused a lot of problems. It was developed by a programmer who worked from home, an hour away from the office, and used a separate code base with unique classes and a different version of the compiler. This was also his first project in 3D, and it became the second-most delayed part of the project after physics. Until the last year of the game, there were significant bugs in the exporter which required time consuming work-arounds Important functionality such as object property exporting also was not delivered until very late in development, preventing designers from implementing gameplay. In the end the exporter was assigned to another programmer and rapidly brought up to usability, but it had already delayed level building significantly.

It has become clear since shipping *Trespasser* that programs like Max are going to serve an increasingly important function in level building in order to create more complex geometry than the current wave of fairly simple custom editors can produce. The mistake *Trespasser* made was assuming that a modeling program like Max could replace all the functions of a custom game editor. A much more pleasant solution for *Trespasser* would have been to use Max only to create world and object geometry, and to assemble it all and implement triggers and enemies within a much-expanded GUIApp. As it was, the extreme difficulty in constructing *Trespasser* levels largely thwarted our final-hour attempts to find the fun gameplay in the systems we had been given.

4. AI problems

The largest problem with the AI system was that its progress was blocked by a lack of dinosaurs with which to test it. The first time a dinosaur

made the transition from a separate test app into the game was in early 1998, and there was significant missing functionality which prevented the completion of visually-important AI behaviors like howling and glaring for much longer. The first quadruped went in around the early summer of 1998, about four months from the then-intended ship date (as it turned out we slipped by about another month).



***Trespasser's* team didn't have enough time to implement dinosaur emotions. [zoom]**

The dinosaur AI was a state-based system where, based on an emotional state, but it became apparent once dinosaurs were working well enough to put into levels that the differences between the activity states were not discrete enough. Dinosaurs were governed by a set of emotions which theoretically would prompt them to pick appropriate responses at any time. However, in practice they would end up oscillating rapidly between many activities, sometimes even literally standing still and twitching as they tried to decide what to do. Making a usable dinosaur required disabling all but one or two of their activities. This allowed aggressive dinosaurs to really be aggressive, but it also meant that the most dinosaurs were as single-minded as the traditional video game monsters we were trying to one-up.

The AI system suffered from the lack of a clear game design. There are two scenes in the Jurassic Park movies which demonstrate quintessential dinosaur gameplay: the scene in JP where the kids hide from raptors in a kitchen, and the one in Lost World where Jeff Goldblum deals with several cautious raptors in the ruins of the town. Both of these scenes rely on dinosaurs which can be fooled by ducking behind objects and which can home in on or be distracted by localized noises. Neither of these two fundamental abilities are actually present in the *Trespasser* dinosaur AI -- the dinosaurs instead have a simpler and more industry-standard detection radius which doubles as sight and hearing and is not blocked by objects in any way. Without a design spec calling for the movie-like behaviors, though, the AI development went in directions which ended up being largely unsuitable for gameplay. This unsuitability wasn't even discovered until a few months before ship, when there was only time to work around it rather than rewrite it.

In addition to more sophisticated senses, the dinosaurs were really lacking an ability to emote. So much of an animal's behavior consists of its various postures and reactions to stimulus that it seems surprising that all *Trespasser* dinosaurs can do is put their heads back to howl, turn to look at their target, bite at an enemy, bend to pick at a carcass, or walk with a weird limp. There was a circling behavior which was cut because the animals became too intent on circling, but there was little else. Given that none of our dinosaur behaviors were pre-animated and therefore didn't require large amount of art time to implement, it should have been simple to add in many expressive behaviors which are typically never seen in games. These, unfortunately, are the kinds of features which either have to be in the specification from day one or which can only be added if there is time remaining after technology is completed and stable.

5. Physics problems

The box model was *Trespasser*'s most significant physics innovation: it was intended to be a complete simulation of any arbitrarily-sized box interacting with a number of other boxes. The approach used in *Trespasser* was what is known as the penalty force method. In incredibly simple terms, when boxes collide, they are allowed to intersect with each other (mathematically), and then they push each other apart until they are no longer intersecting. The penalty force model is generally believed to be an unworkable one by the few other people in the industry attempting real time solids models. *Trespasser*'s physics problems seem to indicate that the rest of the industry was right.

There were several notable flaws with *Trespasser*'s solids model as shipped: it ended up only working well when used with roughly cube-shaped boxes with dimensions between 0.5 and 1 meter, it did not model friction well, it was extremely slow, and it was not free of interpenetration even within the size constraints.

We were aware that physics would be slow, and that ten boxes at once would represent the practical upper limit, but we had not expected so much of that physics overhead to be eaten up by the dinosaur body physics, which used five boxes in the worst cases - head, body, tail, and two feet. Although the dinosaurs physics boxes were supposed to be faster to execute because they did not interact with each other, in practice a common situation of two raptors and the player consumed the entire physics time budget with no room left over to process a tumbling stack of boxes.

The speed was an issue, but the other problems were much more severe. The game design depended on boxes of sizes other than small cubes, and we ended up including many objects outside that safe range anyway. Unfortunately, all objects outside the safe size range, even large cubes, were more prone to reveal the least shippable problem with the physics system: interpenetration.

At best, these interpenetration bugs completely blow the consistency of simulation we were trying to set up, and at worst they make the game

unplayable. If it was not clear before shipping *Trespasser*, it is clear now: no amount of interpenetration is shippable, and preventing it absolutely should be the number one concern of any physics coder.

The size constraints were a major problem for game design. We had been warned from the start that there might be a problem with very small boxes, and thus we avoided specifying too many objects which would need to be small. But we had expected to at least be able to handle somewhat small objects the size of guns and long, thin objects like planks. The last couple weeks of *Trespasser* design bug fixing consisted of frantically removing or making immobile just about every object which did not conform to the very specific object sizes that had proved to be fairly safe.

The friction problem was the main reason we gave up on stacking puzzles - a box on top of another box would more often than not start to vibrate until it had slid off one side or another, and multiple-box stacks were nearly impossible to keep together. It is this very friction problem which has steered many other physics coders away from the penalty force method. Even in an interpenetration-free physics model, the effects of static and dynamic friction are so important to realistic behavior that their absence will compromise the simulation.

Trespasser's dinosaurs and the arm itself were inverse-kinematic (IK) systems controlled by physical models. Although the dinosaurs were originally supposed to be full physically-modeled bipeds whose actual physical model knew how to stand, walk, run, and jump, they ended up being almost identical to the Terra Nova biped model. The dinosaurs were moved through the world using a simplistic physical model which can be thought of as a marble, and the IK system animated the in response to the model's movement. Just like in Terra Nova, the dinosaur legs frequently stretched, bent, and popped as the IK system struggled to handle impossible movements. These movements were incurred by the model itself, which had few realistic bounds to prevent the dinosaur from moving or changing directions in ways that would not be physically possible for a real animal.

The dinosaurs often looked bizarre because of this lack of realistic bounds, but in the case of the arm, the lack of bounds actually contributed to the game's unplayability. The joints of the arm never had realistic limitations put on their rotation or even the distances between joints. A system written by a different programmer sat on top of the underlying arm system and tried to continually make sure it had not moved into an impossible position, but as a separate system it could only be partially successful at best. The arm as shipped would often go almost out of control for a few frames, stretching or spinning in physically unattainable ways. During this time the fragile feeling of connection between the player and their character would be shattered.

However, a properly bounded arm would only have been the first step. The rest of the arm problems were actually a result of bad system design. The arm was intended to be wholly context-insensitive. The fact that there is some context sensitivty causing guns to be held fairly stiffly and away from the body was only a result of some key members of the test staff adding their voices to the cries which had been coming from within the team to fix shooting so that it was remotely possible to hit an intended target. It should have been obvious from the fact that a 2D interface was being used to move in 3D space that a large amount of context-sensitivity was needed. If a truly successful virtual arm is ever to be implemented, simple mouse movements will have to be translated into complicated arm movements based on what is in or near the hand, or it will be as impossible to use as *Trespasser*'s. That there was a conscious decision to avoid context sensitivity is indicative of the larger problems with physics on our project. The physics code was largely written in a vacuum and tested in separate applications and non-representational levels and not enough of an attempt was made to step back and analyze it from a player's perspective and make it into a system which was designed to support gameplay in every way.

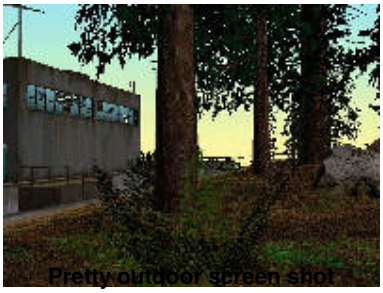
The Value of Hindsight

Why was *Trespasser* designed as a software-only product when the industry was beginning to embrace hardware in a big way? Why did it enter full production with only a prose walkthrough for a design spec? Why did the AI system need a major shift in direction in the last few months of production, and why did physics code which is barely usable actually ship? The answer to all these questions is that *Trespasser* was a project with management problems at all levels.

The word "management" is almost a profanity to some developers with its connotations of hierarchy and structure, so at odds to the often anarchic side of game making. Some major companies abhor the notion so much they have even done away with titles altogether. However, after the *Trespasser* experience it has become clear that just as there are no successful anarchic world governments there can not be any successful development teams without management.

Trespasser was developed for a Hollywood company, and if there's one thing Hollywood is good at it is wholeheartedly embracing the notion of hierarchical structures. However, this does not mean that Hollywood knows how to manage successfully. Instead it is famous for its tales of producers on power trips ruining projects and careers left and right as it pleases them. Perhaps Trespasser fell prey to this Hollywood curse. At the very least, it suffered from being an innovative, technologically ambitious project being developed at a company which had not yet gained institutional experience in game management by publishing significant less-ambitious projects.

The lack of skill and experience at the highest level meant that the *Trespasser* project itself was allowed to run almost entirely out of control. Many developers have had experiences with upper management taking too active of a hand in the production of individual products and frustrating teams with unworkable ideas and arbitrary demands, and being free of upper management interference is their dream. However, as it turned out for *Trespasser*, having no management influence above the producer level was just as much of a problem.



Pretty outdoor screen shot from *Trespasser*. [zoom]

Trespasser's team management was largely made up of people who had not had specific experience with running complicated projects and large numbers of employees before, or who had not managed a game project before. Computer games are one of the most difficult project management tasks possible, consisting as they do of a careful balance between art and engineering, between the desire to innovate and the need to deliver a product in some sort of timely fashion. Managing the types of people who make computer games is also a tremendous test of interpersonal skills, especially when, as on the *Trespasser* team, many of them have never worked in the industry before and lack of familiarity with standard game development practices. All of *Trespasser*'s managers, from the top of the team on down, needed mentoring and support from above in order to develop into the kinds of people who could handle the many complexities of such a complicated project, but never received it.

One of the surest signs of management problems is a lack of communication, and *Trespasser* was rife with it. The design document is a principal example of this communication problem, as has been mentioned above.Also significant was a lack of regular and useful team meetings. Most projects bring all their members together on a regular basis in order to share information, air concerns, and generally improve the atmosphere of teamwork. *Trespasser*'s team meetings happened erratically at best, going from unnecessarily often - as frequently as three times a week - to weeks or even months with no meetings at all. When the meetings did happen, they often served more to demotivate rather than motivate , as the project was continuously behind and the team leadership was not particularly successful at being positive about the delays. Towards the end of the project, the team largely came to dread them rather than look forward to meetings of any kind, such that by the time the project finally entered beta, no attempt was even made to have team bug meetings, which greatly contributed to the stress and frustration in the final days.

The general awkwardness of team relations was exacerbated by having a key part of the whole project - the physics - written by the project leader. It used to be that it was possible to both run a project and contribute key work to it, but with a game as complicated and a team as large as *Trespasser*, it was not reasonable to expect that this would be possible. Beyond being a nigh-impossible task, it also put the department heads of the project, especially the lead programmer, in an unenviable and difficult position. When the physics code was continuously delayed and unsatisfactory, there was no easy way to take action to fix those problems. No lead programmer should be expected to tell their boss that he had better get his work done or take some drastic action to get the project back on schedule. Further complicating matters was the lack of upper management support - although on several occasions the team did attempt to go to higher levels to have the lack of progress addressed, the company's higher-ups were by turns unable and unwilling to take the necessary steps. Perhaps worst of all were the personality conflicts which arose - after a while, it became impossible to even attempt to raise concerns about the physics code without in a constructive rather than confrontational manner. Many on the team began to go out of their way to avoid dealing with the issue at all, but this only allowed the problems to continue growing. By the end of the project, the team was faced with the difficult task of having to take what we had, no matter how unsatisfactory, and try to get it into a stable enough state to ship.

It is unfortunate that even though the *Trespasser* team was made of some of the most brilliant programmers, artists, and designers in the industry, the game we ended up shipping was at best a deeply flawed beta. In the end, when most of the team were fed up with the whole project and exhausted from what amounted to more than a year in ship mode, it became an issue of ship or possibly never ship. The only way *Trespasser* was going to become the game that we had promised to the world was to take dramatic steps to address the management issues which existed, and few companies in the industry would have had the commitment and vision to take those steps. With an immense amount of effort, the game did manage to meet its last possible ship date despite the difficult circumstances, but it is hard to imagine what we could have accomplished if we had been brought together and guided with a strong vision and strong plan.

It has been almost half a year since *Trespasser* shipped. In that time it has gone from a gigantic Christmas letdown to an occasionally-referenced joke. The team has mostly disintegrated, some quitting, some let go, and those remaining distributed across several different projects. The engine is effectively dead, with the new DWI motto being "licensed technology," probably a good idea for the company but pretty disheartening for the engineers who created last year's most innovative, if not quite best, engine. A group of fans on the net have proclaimed themselves the *Trespasser* Hacking Society and have taken up the lunatic task of trying to figure out how to do a mod for an engine which was barely usable with its in-house tools.

That *Trespasser* shipped at all is a testament to the strength of the individual members of its team. In looking back at my own experience on *Trespasser*, I find that I have learned a lot about game development, and am already putting that knowledge to work. Although *Trespasser* the game does not fulfill the many high hopes I had for it or even my base expectations for a piece of shipped software, I am happy with the work I put into it. It is my sincere wish that every other person who contributed to the massive effort is equally proud of their work, and that sometime in the future we all have our chance to make a major project which succeeds where *Trespasser* failed.

Richard Wyckoff was a designer on* Trespasser *who has worked previously at Looking Glass Technologies in small roles on projects like* Flight Unlimited *and* Terra Nova*. He is currently applying what he learned from* Trespasser *on an unannounced project at* Knowledge Adventure*. He can be reached at richw@loonygames.com.

Trespasser
DreamWorks Interactive
Los Angeles, California

www.dreamworksgames.com

Team Size: (the team included all these people at various points) Producer: Seamus Blackley. Associate Producer/Audio Producer: Brady Bell. Engineers: Andrew Grant, Paul Keet, Mark Langerak, Mike Mounier, Scott Peter, Greg Stull, Rob Wyatt. Additional Programming: Richard Benson, Steve Herndon, Brandon Lee, Kevin Sherill, Charlie Wallace. Designers: Chris Cross, Austin Grossman, Alan Hickey, Brian Reed, Richard Wyckoff. Artists: Jenny Hansen, Terry Izumi, Jay Jang, Lonnie Kraatz, Kyle McKisic, Rolf Mohr, Brian Moore, Antonia Olzsowka, Marta Recio, Phil Salas. Additional Art: Diane Chang, Sean Connor, George Edwards, Wei Ho, Daniel Wong, James Wong. Production Assistance/Asset Management: Jon Galvan, Greg Hillegas. Marketing: Rich Flier. Marketing Assistance: Amy Nabi.

Release Date: October 1998

Estimated Budget: $6-7 million

Time in development: 32-36 months

Typical workstation: Pentium II 266 MHz, 128-256MB RAM.

Critical Applications: 3D Studio Max 1.2, 2.5, Photoshop 4.0, Microsoft Visual C++ 6.0, Visual SourceSafe 5.0

Intended platform: Windows 95/98/NT

Return to the full version of this article