Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Crowdbrush: Interactive Authoring of Real-time Crowd Scenes

Branislav Ulicny, Pablo de Heras Ciechomski and Daniel Thalmann

Virtual Reality Lab, EPFL, CH-1015 Lausanne, Switzerland

Abstract

Recent advances in computer graphics techniques and increasing power of graphics hardware made it possible to display and animate large crowds in real-time. Most of the research efforts have been directed towards improving rendering or behavior control; the question how to author crowd scenes in an efficient way is usually not addressed. We introduce a novel approach to create complex scenes involving thousands of animated individuals in a simple and intuitive way. By employing a brush metaphor, analogous to the tools used in image manipulation programs, we can distribute, modify and control crowd members in real-time with immediate visual feedback. We define concepts of operators and instance properties that allow to create and manage variety in populations of virtual humans. An efficient technique allowing to render up to several thousands of fully three-dimensional polygonal characters with keyframed animations at interactive framerates is presented. The potential of our approach is demonstrated by authoring a scenario of a virtual audience in a theater and a scenario of a pedestrian crowd in a city.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Picture/Image Generation]: Display algorithms I.3.4 [Graphics Utilities]: Graphics editors I.3.7 [Three-Dimensional Graphics and Realism]: Animation

1. Introduction

With current consumer-grade personal computers it is possible to display 3D virtual scenes with thousands of animated individual entities at interactive framerates using different techniques as animated impostors [TLC02] or point-based rendering [WS02].

When increasing the number of involved individuals it is becoming more difficult to create unique and varied content of scenarios with large numbers of entities. If we want to create or modify features of every single individual one by one, it will soon become too laborious. If, on the other hand, we apply a set of features (either uniform, or patterned) to many individuals at once, it could create unwanted artefacts on a larger scale, resulting in an "army-like" appearance with too uniform, or periodic distributions of individuals or characteristics. Use of random distributions can alleviate such problems; however, it can be very difficult to capture the desired constraints into a set of mathematical equations, especially considering integration into common art production pipelines.

The challenge is how to create complex scenes resembling a variety-rich look of the real world. Here we understand complexity analogous to a notion of complexity of patterns generated by cellular automata as in [Wol02]: not uniform, not periodical, nor just fully random.

Bottom-up approaches, such as local rule based flocking [Rey87] can create such complexity, however they are difficult to control if we want to achieve particular end configurations (how to set local rules to get a global result). In the recent work Anderson et al. [AMC03] achieved interesting results for a particular case of constrained flocking animation. Nevertheless, the algorithm can get very costly when increasing the number of entities and simulation time.

Major 3D content creation packages used by the media industry now offer tools to improve working with a large number of virtual characters [CS04, SIB04]. The most advanced crowd animation system for non real-time productions is Massive; used to create battle scenes in the Lord of the Rings movie trilogy [Mas03]. The production of massively populated scenes is still in the majority of cases a lengthy and

Figure 1: *Crowdbrush application.*

manual-intervention intensive process, operating mostly in a non-real-time mode. A main part of the design process generally uses simple proxy objects that are turned into full characters only in the final stage of production. Manipulations of scenes and characters usually involve complex sequences of menu selections. An interesting approach to add sets of objects (as clouds, trees, flowers, or buildings) to the scene is used in Maya Paint Effects [MAY04], where a designer can paint pseudo-3D objects in the scene using 2D brush strokes. Such objects are not fully integrated into the 3D scene: they are rendered in a special buffer with separate shading and are further composited into the final image as a Z-buffer based postprocess.

Our approach is to give full creative power to designers using metaphors of artistic tools, operating on a two-dimensional canvas familiar from image manipulation programs working in WYSIWYG (What You See Is What You Get) mode, with a real-time view of the authored scene. The advantages of immediate feedback, intuitive interface and familiarity allow to better express the artist's vision and can also lead to an increase in productivity.

The structure of the paper is as follows: first, we present the overall design of our system discussing the requirements needed for interactive authoring, followed by a detailed account of the concept of brushes. Next, we describe crowd behavior, rendering and animation engines, and finally we present the results, where we use a prototype of the crowd brush application to create a scene of a virtual audience in a reconstruction of an ancient theatre and a scene of a pedestrian crowd in a virtual city.

2. System overview

Our goal is to create a system that would allow authoring of freely navigable real-time 3D scenes, composed of a large number of varied animated individuals in a virtual environment. The authoring should by simple and intuitive, usable by non-programmers.

We take inspiration from the domain of image and word processing, where most of the applications use WYSIWYG approaches with immediate feedback of the resulting manipulations. In computer graphics such an approach was used, for example, for direct painting on models [HH90], [KMM*02], sketching of 3D models out of 2D drawings [ZHH96], creating facial expressions [PHL*98], or to paint 2D images with a virtual brush [XLTP03]. A paint metaphor was used to design layouts of objects for information visualization [Tob03], and it was shown to be efficient to set values of toggle switches [Bau98]. Salesin and Barzel [SB93] presented the adjustable tools metaphor, where each operation is bundled with its collection of attributes to define a single tool.

The idea is simple: the designer manipulates virtual tools, working in a two-dimensional screen space, with a mouse and a keyboard. These tools then affect the corresponding objects in a three-dimensional world space (see Figure 1). Different tools have different visualizations and perform dif-

ferent effects on the scene including creation and deletion of crowd members, changing of their appearances, triggering of various animations, setting of higher-level behavioral parameters, setting waypoints for displacement of the crowd, or sending of events to a behavior subsystem.

We briefly experimented with a fully three-dimensional interface, where tools existed in a 3D world space. Nevertheless it appeared to be not very practical, at least not when using standard input devices operating in two dimensions as a mouse or a trackball. The usability of a 3D interface could be improved using some truly 3D input devices such as a spaceball, a 3D mouse, or magnetic sensors. However, it would limit the number of potential users as such devices are not common.

In order to create an authoring tool as outlined above, the system, on which it will operate, should fulfil the following requirements:

Individuality: The system must allow for each individual instance to have a set of attributes, and not share them among many individuals, as they can potentially have unique values, and can be individually selectable.

Responsiveness: It must be fast enough for real-time editing to allow for an immediate feedback loop. Therefore, it must not involve lengthy preprocessing stages (at least for features that are target for authoring) and also the system's responses to changes must be fast.

The requirements are close to those of any interactive application. Indeed, boundaries between authoring and interaction are getting more and more fuzzy. A recent trend in game development is to use the same application for part of the authoring and actual gameplay (or to integrate play-test ability in authoring tools) [RWS04, GBO04]. The player interaction is, then, the subset of possible interactions for a designer. Such designs came from the necessity of constantly switching between "play" and "create" modes while developing the game in order to increase productivity.

Figure 2 shows an overview of the system design. The user controls the application using a mouse and a keyboard. The mouse moves the visual representation of the brush tool (we used an icon of a spray can) on the screen, with the mouse buttons triggering different actions either on rendering or behavior subsystems. The keyboard selects different brushes, sets their parameters and switches between "navigate" and "paint" modes. In the "navigate" mode, the mouse controls position and orientation of the camera. In the "paint" mode, the camera control is suspended and different areas on screen are selected depending on the pressed mouse button. The selection areas can be, in principle, any arbitrary 2D shape; in the current implementation, the selection can be a single point, a circle, or a rectangle. This area is then further processed by the brush according to its particular configuration (see next section).

The integration of the interface, behavior, and rendering subsystems is done using VHD++, a component based framework for creating real-time VR systems [PPM*02].

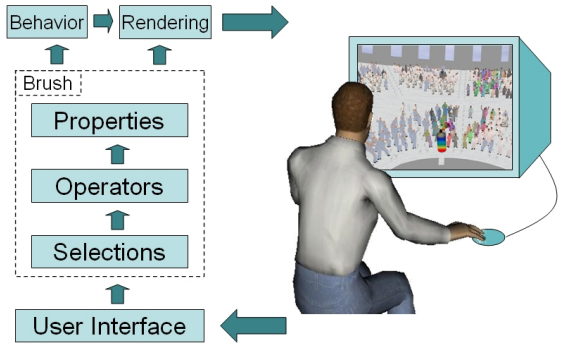

Figure 2: *Overview of the system design.*

3. Brushes

Brushes are tools with visual representation on the screen that affect crowd members in different manners: for example, the brush can create new individuals in the scene, or it can change their appearances or behaviors. We selected visualizations of the brushes to intuitively hint on function. For example, the creation brush has an icon of a human, the orientation brush has an icon of a compass, the deletion brush has an icon of a crossed over human, and so on.

The brush is processed in three stages. First, a selection of the affected area in the 2D screen space is performed according to a triggered mouse button, with subsequent picking of the entities in the 3D world space. Then, the operator will modify the manner of execution of the brush in the selected area. Finally, the brush will change the values of the instance properties for the affected individuals. In case of the creation brush, it will create new population members; for the event brush it will send events to a behavior system and for the path brush it will add a new waypoint to a current path.

Values of the scalar brush parameters can be controlled by a keyboard; for example, sizes of waypoints for a path brush, or speed of walk for an event brush are increased and decreased by pressing '+' and '-' keys, affecting a corresponding active brush. Vector brush parameters can be set by direction of the strokes, so that for example, a direction of movement depends on the temporal order of drawing waypoints.

Selections are defined in screen-space. A selection can be a single point at the location of a cursor, or an area around a cursor. If the selection is a single point, picking in the 3D world is performed by computing the intersection of a line segment with the scene. If the selection is an area, picking is performed on a random sample of points from that area, following a "spray" metaphor. The size of the

Figure 3: *Creation brush with random operator.*

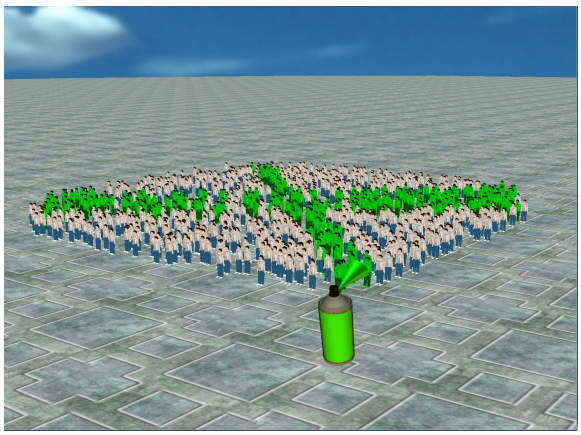

Figure 4: *Colour brush with uniform operator.*

selected area in the world space changes with the level of zoom into the 3D scene. This provides an intuitive control of focus: if we want to work on a large part of the crowd, we zoom out of the 3D view to see a desired part or whole of the crowd. If we want to focus on a smaller group, or individual, we zoom in. Thus features that are observable from a particular zoom level are also editable at this level.

Operators define how selection will be affected. For example, a stroke of the creation brush with the random operator would create a random mix of entities (see Figure 3); a stroke of the uniform colour brush would set colours of affected individuals to the same value, as shown in Figure 4; and a stroke of the orientation brush with the gradient operator would let individuals turn in the direction of the gradient (see Figure 5). In the current implementation, the distribution of values for random operator is uniform; it could be extended to use different distributions, for example, by drawing 2D distribution curves in screen space.

Instance properties are non-sharable attributes, giving uniqueness to every individual member of the crowd. Instance properties encapsulate lower-level features influencing both appearance and animations of virtual humans. Spatial configuration is qualified by properties of position and orientation. Appearance is influenced by properties of colour, texture, material and scale. Execution of an animation action is determined by animation selection, shift, and speed properties. High-level features can use a combination of several low-level features accessed through their properties. For example, a particular emotional state would set animations from a predefined set with some specific speed, or clothing style would select a set of appropriate textures and colours for different body parts. The set of values of all instance properties fully defines the configuration of the crowd members, and is used to save and restore the state of the scenario during and across the authoring sessions.

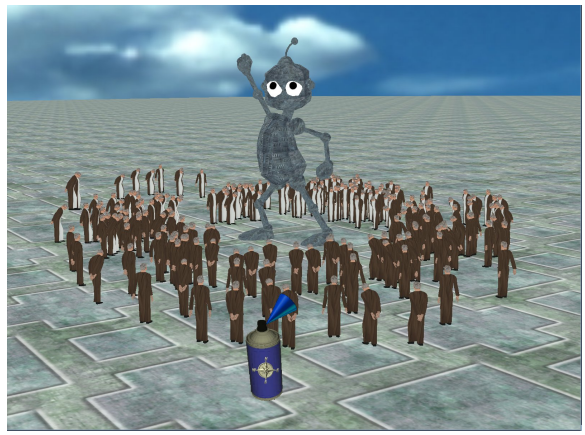

Figure 5: *Orientation brush with gradient operator (Hugo model by Laurence Boissieux ©INRIA 2003).*

4. Behavior

A behavior is not the focus of this article. Nevertheless, as more complex scenarios require some sort of behavior simulation [MT01], we implemented a simple spatial displacement of the humans and a rule based behavior engine [UT02] to handle more autonomous characters. We opted for a simple reactive rules system as we needed fast execution for up to several thousands of agents. Behavior rules can react to both internal and external events to the agents, triggering sequences of actions as displacement or animations.

Agents can move around the environment following paths which are sequences of places with potentially variable sizes. The list of all places and paths is stored by the environment manager which responds to queries from a rule system. Positions of the agents between waypoints are interpolated taking into account their desired speed and actual simulation update rate. Along the way, agents play looping

keyframed animations of different walking styles or running depending on the displacement speed.

The behavioral rules and the environment database are editable in run-time using rules and environment editors connected with the behavior engine.

We added also a simple collision avoidance system based on a force-based model [HM95]. In order to improve efficiency of the collision avoidance, spatial queries need be optimized; otherwise collision detection would become a performance bottleneck much sooner than rendering. We used a simple bin-lattice space subdivision to minimize the number of collision queries [Rey00].

The brush metaphor is integrated with behavior simulation in a direct mode by sending events to the rule system and the environment manager. The spray can be then used, for example, to send events to activate behavior rules which send an agent to a path, or to add environment objects used for navigation and behavior control. The brushes can also be used to control the behaviors in a non-direct mode, where the desired agents are first tagged by spraying with a tagging brush. Later, this tagging information is used to selectively trigger different behavioral rules for different agents, allowing for multiple actions to happen in different parts of the scene at the same time.

5. Rendering

When designing our rendering system, we set out to achieve several goals following the requirements introduced in Section 2. The first condition was to be able to render and manage simultaneously at least one thousand humans while keeping an interactive framerate. Second, we needed a variety of appearance and behavior in the crowd while constraining memory usage. We should also be able to render the humans with lighting. In order to keep the data pipeline from model to rendering engine smooth, we needed fast and robust pre-processing of data.

Existing Approaches: We considered several already existing crowd rendering approaches. Tecchia et al. [TLC02] used billboards or impostors for crowd rendering. The main advantage is that impostors are very light-weight to render once they are in the memory of the graphics card. The method requires building of all animations from all possible camera angles, and storing these pictures in a texture. One such texture can hold one frame of animation in very low resolution billboards, where every individual sub-frame is about 16 pixels tall and wide. This can give good visuals since it is basically an image based rendering approach, so even pictures of real humans could be incorporated. However, zooming on these billboards will produce aliasing artifacts, due to the fact that the images on the billboards have to be small to fit in the graphics cards texture memory. This makes billboarded humans a good approach for far-away humans that do not need detailed views.

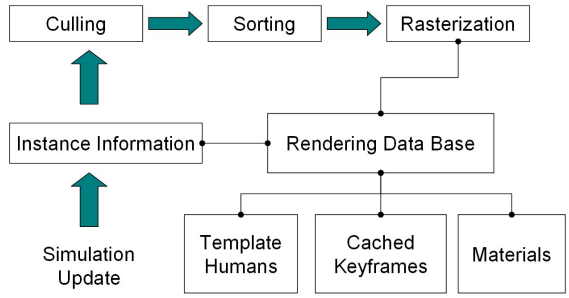

Figure 6: *Rendering pipeline.*

Another approach which unifies image based and polygonal rendering is found in [WS02]. They create view-dependant octree representations of every keyframe of animation, where nodes store information about whether it is a polygon or a point. These representations are also able to interpolate linearly from one tree to another so that in-between frames can be calculated. When the viewer is at a long distance, the human is rendered using point rendering, when zoomed in using polygonal techniques and when in between a mix of the two. It does take large amounts of data per keyframe and needs long pre-processing times because of its precise nature, but also gives near perfect interpolation between detail levels without "popping" artifacts which otherwise occur if one uses discrete detail levels.

The third alternative was to use vertex shaders to deform a conventional mesh using a skeleton. The disadvantage would be that the pipeline would be constrained to shaders and every interaction such as lighting, shadows and other standard effects, would then have to be programmed with shaders.

The method of vertex tweening was used in the game Quake [Qua96] for a smaller number of virtual humans. In this approach, fully precomputed meshes are exported by the 3D designing software since dynamic skinning was not feasible at the time of game release. The main disadvantage of this approach is complex and tedious data preparation and exporting pipeline.

Our main idea is to store complete meshes in OpenGL display lists and then carefully sorting them taking cache coherency into account while rendering. This results in a method which has little or no processing on the CPU because precomputed meshes are stored on the graphics card. The only consideration we had to do was to manage the rendering, so that humans that have the same materials, meshes or animations are rendered at the same time, since graphics cards need to be kept cache coherent to perform well. Figure 6 shows an overview of our rendering pipeline.

Data Processing: We chose to use skeletal models with deformed meshes as the basis of our animation system. Our crowd consists of a large number of individuals that are cloned from a smaller number of template humans. All in-

stances of the same template have the same mesh and skeleton.

One of the important considerations we had to take into account was the ease of data production using already existing 3D content production tools. We wrote a mesh and skeleton exporter for 3ds max using MaxScript [3DS04]. The mesh itself consists of vertices and triangles. The exported triangles have associated information such as texture coordinates, materials and skeleton bindings. The next step in the data-production pipeline is exporting a reference mesh and a reference posture describing the initial animation for applying skeletal transforms using the Bones Pro 3 plug-in [BP304]. This allows exporting of any animation for a particular skeleton. The animation is defined as a list of bones in the hierarchy with translation and rotation for every sampled keyframe.

For importing the model into the engine we have to first unpack it into complete meshes. For every animation we define a running time in seconds and every second is sampled thirty two times to preserve fluidity of animation. Sampling is done on skeletal keyframes using spherical interpolation on rotation quaternions, and using a linear interpolation for the translations. The sampled skeleton is applied to the mesh and deforms it using conventional skinning methods [Lan99]. When the deformed mesh has been computed, the result is stored into a display list [SWND03]. If a character has more than one material, which in our case is more than one texture, the character is divided into several display lists, each with its own material. This allows us to exchange textures for different parts of the body in run-time.

Rendering Management: For management of the scene we use the OpenSceneGraph 3D graphics toolkit [OSG04], where the crowd renderer is integrated as a custom node using OpenGL rendering [SWND03].

After all characters have been loaded into memory, they are kept in a database called the manager. The manager stores template humans and also keeps record of all existing instances. A template human consists of a set of animations, which in turn keep lists of display lists and textures. An instance keeps hold of the properties for the human giving uniqueness to every individual (see Section 3). The instance properties are: animation index, time inside animation, speed of animation, mesh colour, mesh scale, material settings (if there are many possible, for example, different textures are possible for the torso, face, and so on) and a transformation matrix for translation and rotation.

Before the start of rendering, all initial instances of the template humans are registered and their properties are set. When the rendering loop starts, the manager is called to render all instances simultaneously. Rendering is optimized to minimize state changes by sorting instances by vertex buffers and textures. The most simple approach is to render by template, then by material set and finally by instance. Just before the display list is called, the instance properties are applied to give the impression of variety.

The approximate data size of unpacked models ready for rendering is computed in the following manner. Each triangle has three vertices, three normals and three texture coordinates, all together with the size of approximately one hundred bytes. For each frame, the size requirement in bytes is one hundred times the number of triangles. An average human that consists of one thousand triangles requires around one hundred kilobytes per keyframe. Since we sample the animation at thirty two frames per second, this amounts to approximately 3.2 megabytes per second.

The benefit of using display lists is that we can take advantage of the full OpenGL pipeline without any changes to lighting. The characters look well from all camera directions and zooms, as opposed to billboards which are usually low resolution and need pre-processing for all different camera angles. The memory usage is lower than with billboards, however more polygons need to be rendered. The data storage requirements on disk are very low, since we store only a basic mesh and animations of the skeleton.

6. Results

We tested usability and responsiveness of the authoring application on two very common situations involving a large number of humans: we created scenarios of a virtual audience (see Figure 1) and the scenario of a pedestrian crowd (see Figure 8).

Virtual audience: In the first scenario the tasks are, given existing models of a theatre and template humans, to fill the building with an audience and distribute the animations according to the desire of the designer. The specific scene has to be constructed according to a typical distribution of social classes and behaviors in antique theaters extracted from the historical literature. We used four different template humans, all with around one thousand triangles. The theater model has around fourteen thousand triangles.

In order to facilitate positioning of the humans, we created a grid of valid positions in the theatre following the distribution of the seats. The creation brush was then restricted to operate only on this grid, instead of free picking. Using the grid we can position the humans without caring about collisions, for example, if two humans happen to materialize very close to each other. The correct position and orientation of the audience is automatic, in this way, the same scene will have a certain expected behavior when you interact with it, much in the same way as in a paint program, where pixel positions are placed in a grid.

Besides creation of the scene, we used brushes also for interaction with the audience. As soon as humans start to appear in the scene, we could change their behaviors by using the "emotion" spray that makes the human play different kinds of animations from predefined sets, for example,

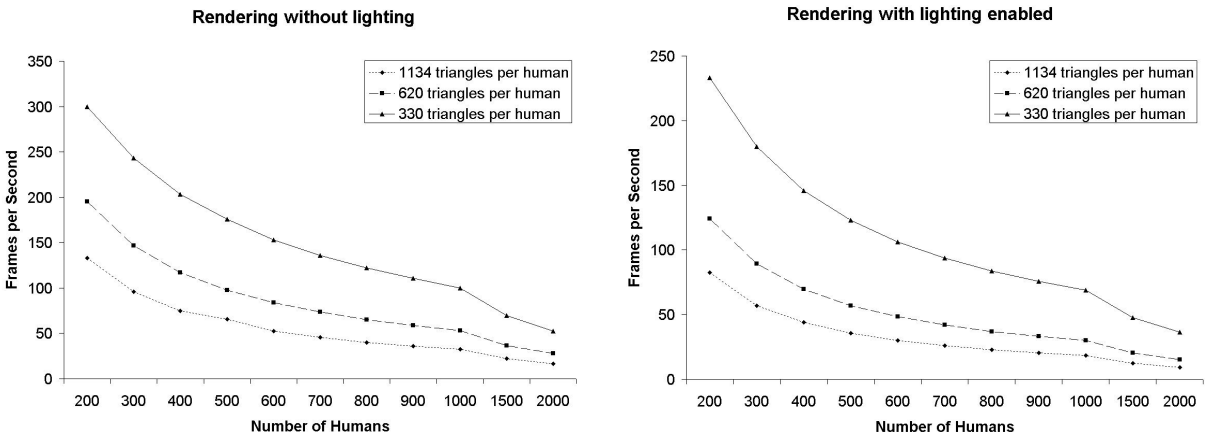

Figure 7: *Performance statistics.*

Figure 8: *Crowd following paths defined by path brush.*

happy animations or sad animations. The particular state of the audience can be saved and then restored (see Section 3), so that, for example, different reactions to different actors are prepared and then replayed when needed.

Virtual pedestrians: In the second scenario the tasks are, given models of a virtual city and template humans, to distribute pedestrians to desired locations in the city, to set up network of paths among specified locations and finally to let selected pedestrians follow these paths. In this scenario we used eight different template humans with around one and a half thousand triangles each; the model of the city has around sixteen thousand triangles.

In comparison with the previous scenario, the placement of the newly created humans was not restricted to a grid; creation brush could add pedestrians at any desired location in the 3D world. The only constraint was that for a drag motion of the brush, the new human had to be at least at some minimal distance from the previously created one. Even this constraint is not completely necessary as force based collision avoidance will take care of not letting humans overlap. However, we found that imposing it was giving more aesthetically pleasing results.

In addition to creation of humans, we used the brush to add sequences of places to the environment. Our environment model allows to have places with different sizes which allows, for example, to channel a flow of the crowd to wider or more narrow streets. The paths can be of three types depending on the desired behavior on the boundaries of the path. At the end of a simple path the agent stops; a cyclic path lets the agent again move towards the first waypoint; and a teleport path sets the position of the agent to the first waypoint. Changes of the place size and designation of paths from active set of places is in the current implementation done via keyboard.

Once the paths were defined, we could send events to selected pedestrians by spraying them with the event brush. The crowd behavior system then responds to received events and let the agents move following the paths. Furthermore, we can affect speed of movement of selected agents by spraying them with the event brush, now sending a different type of event. Then, the event triggers a rule which changes speed of locomotion and the associated animation, for example, to a running motion instead of a walking one.

For both scenarios we kept interactive framerates at all times, for example, when adding and removing humans from the scene, or when modifying attributes, such as animation, colour or orientation. When increasing the number of humans in the audience, performance was dropping, yet even with full capacity of the theatre (around 700 places), an acceptable rate of 27 frames per second with lighting enabled was maintained. For the pedestrian crowd, the performance was acceptable for up to around 500 agents simultaneously on the screen, which is less than in the first scenario due

to the higher complexity of models and also because of the overhead of a behavior simulation. Because of the responsiveness of the crowd engine and therefore also of the interface, manipulation of the scene was immediate and intuitive, since changes appeared on the fly.

Rendering: In order to measure the performance of the rendering engine, we ran several tests, where we varied the number of humans and the number of triangles per human. The performance tests were run on a Pentium4 2.2 GHz machine with an NVIDIA Quadro4 graphics card. We performed two sets of tests, depending on whether lighting was enabled or not. As can be seen in Figure 7, the crowd composed of thousand characters with 1134 triangles was rendered at around 33 frames per second without lighting. Using simpler characters yields a significant increase in the performance: a crowd of two thousand humans with 330 triangles was rendered at 53 frames per second. The lighting brings overhead of around twenty to eighty percent depending on the triangle count.

7. Discussion

Our authoring approach can work in conjunction with different crowd rendering systems if they fulfil the requirements defined in Section 2. It would only be necessary, then, to define an instance properties layer, abstracting implementation details to interface a particular crowd rendering engine with brushes. CrowdBrush is also complementary to works on crowd behavior simulation, where it can be used, for example, to distribute or modify attributes and features over population of crowd agents, to paint relationships among the crowd members, or to send events to their behavior engines (see Section 6).

The main limitation of our spatially oriented brush metaphor is a weak control of time related aspects of scenarios. We address this issue by incorporating behavior rules engine that is complementary to brushes (see Section 4). Furthermore, as the Crowdbrush is integrated into the VHD++ framework, the other authoring methods provided by the platform as Python scripts can be used in conjunction with the brush metaphor.

The advantage of our rendering approach is that polygonal models are fully integrated into the triangle processing pipeline, allowing for lighting (see Figure 9) or even more complex shading effects. For example, it was straightforward to allow the crowdbrush application to render a crowd with specular highlighting (see Figure 10) or cartoon shading (see Figure 11) effects using an existing library of effects [OSG04]. As well, our rendering pipeline is using only standard OpenGL features, thus being independent of any hardware specific extension.

Furthermore, because all characters are fully animatable using a skeleton, it would be possible to integrate our crowd with some more complex motion control methods beyond simple keyframes. For example, if the scenario would require it, some particular members of the crowd could become animated by procedural walk generation, inverse kinematics, or motion graphs.

Our rendering engine is flexible to go beyond humanoid characters. We successfuly imported fully skinned and animated characters and objects from several computer games, such as four legged animals, insects, helicopters or a forklift.

The main limitation of our rendering approach is a relatively large amount of memory required in case more templates and more animations are needed to increase the variety of a crowd. However, this problem is common to all current methods for real-time rendering of large crowds as impostors or point-based rendering, which have memory requirements several times higher.

Figure 9: *Crowd rendered with coloured spotlight.*

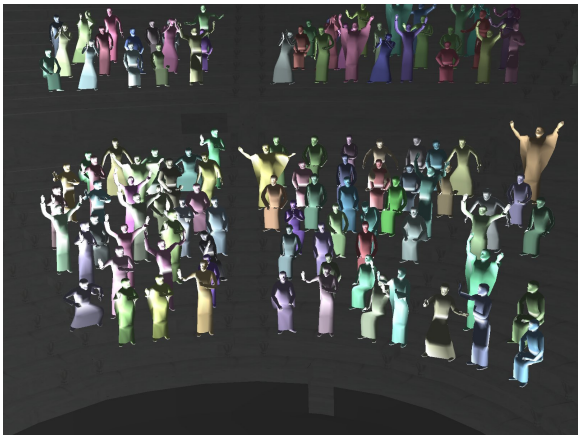

Figure 10: *Crowd rendered with specular highlights shading.*

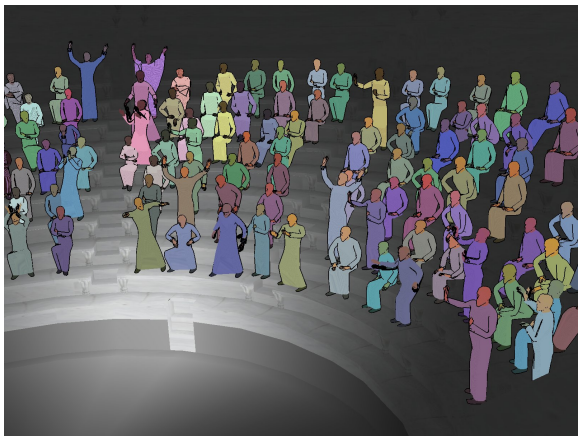

Figure 11: *Crowd rendered with cartoon shading.*

8. Conclusions and Future Work

We presented a novel technique for authoring real-time crowd scenes in conjunction with a management of large real-time rendered crowds.

The benefits of our approach are twofold: an intuitive and familiar 2D interface (WYSIWYG) for authoring and interacting with crowds of humans in real-time; and the ability to render a large number of fully skinned and animated 3D characters in real-time, in a standard way that is easily integrated into an existing triangle processing pipeline such as OpenGL.

We tested our application on authoring the scenario of a virtual audience, as seen in Figure 1, and on authoring of a pedestrian crowd (see Figure 8). We concluded that it was easy to configure the scenes according to our wishes.

Our initial tests with an artist user confirm benefits of the intuitive interface: the artist was able to create crowd scenes using the CrowdBrush after a few minutes instruction, without the need of lengthy tutorial and detailed user manual. The user was provided only with a one page quick reference card displaying keyboard shortcuts associated with different brushes.

There are several directions for future work. We will concentrate on extending the possible number of handled entities and on improving behavior repertoire. Our preliminary tests with level-of-details show a promising increase in performance, or conversely an increase in number of humans while keeping interactive framerates. Also a billboarding approach for far-away humans is being incorporated, similar to the one in [TLC02] and it will be interesting to see how these two will work in conjunction.

For the user interface, brush parameters specification can be improved by having dedicated widgets allowing to modify the properties of the active brush. For example, the actual colour used by a colour brush could be selected from a standard colour palette dialog, or a template for a creation brush could be selected from a swatch of available virtual humans.

9. Acknowledgements

We are grateful to Mireille Clavien and Rachel de Bondeli for the design of virtual humans and scenes. This work has been supported by the Swiss National Research Foundation and the Federal Office for Education and Science in the framework of the European project ERATO.

References

[3DS04] 3ds max, 2004. http://www.discreet.com/3dsmax.

[AMC03] ANDERSON M., MCDANIEL E., CHENNEY S.: Constrained animation of flocks. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)* (2003), pp. 286–297.

[Bau98] BAUDISCH P.: Don't click, paint! Using toggle maps to manipulate sets of toggle switches. In *Proc. UIST '98* (1998), pp. 65–66.

[BP304] Bones Pro 3, 2004. http://www.digimation.com.

[CS04] Character Studio, 2004. http://www.discreet.com/products/cs.

[GBO04] Gamebryo, game engine, 2004. http://www.ndl.com.

[HH90] HANRAHAN P., HAEBERLI P. E.: Direct WYSIWYG painting and texturing on 3D shapes. In *Proc. SIGGRAPH '90* (1990), pp. 215–223.

[HM95] HELBING D., MOLNAR P.: Social force model for pedestrian dynamics. *Phys. Rev. E 51* (1995), 4282–4286.

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: Drawing strokes directly on 3D models. In *Proc. SIGGRAPH'02* (2002), pp. 755–762.

[Lan99] LANDER J.: Over my dead, polygonal body. *Game Developer Magazine* (May 1999), 1–4.

[Mas03] Massive, crowd animation software for visual effects, 2003. http://www.massivesoftware.com.

[MAY04] Maya, 2004. http://www.alias.com/maya.

[MT01] MUSSE S. R., THALMANN D.: A hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 2 (April-June 2001), 152–164.

[OSG04] OpenSceneGraph, 2004. http://www.openscenegraph.org.

[PHL*98] PIGHIN F., HECKER J., LISCHINSKI D., SZELISKI R., SALESIN D. H.: Synthesizing realistic facial expressions from photographs. In *Proc. SIGGRAPH '98* (1998).

[PPM*02] PONDER M., PAPAGIANNAKIS G., MOLET T., MAGNENAT-THALMANN N., THALMANN D.: VHD++ real-time development framework architecture: Building flexible and extendible VR/AR systems with reusable components. In *Proc. Computer Graphics International 2002* (2002).

[Qua96] Quake, game homepage, 1996. http://www.idsoftware.com/games/quake/quake.

[Rey87] REYNOLDS C. W.: Flocks, herds, and schools: A distributed behavioral model. In *Proc. SIGGRAPH '87* (1987), pp. 25–34.

[Rey00] REYNOLDS C. W.: Interaction with groups of autonomous characters. In *Proc. Game Developpers Conference '00* (2000), pp. 449–460.

[RWS04] RenderWare Studio, game development platform, 2004. http://www.renderware.com/renderwarestudio.html.

[SB93] SALESIN D., BARZEL R.: Adjustable tools: An object-oriented interaction metaphor. *ACM Transactions on Graphics 12*, 1 (1993), 103–107.

[SIB04] Softimage XSI Behavior, 2004. http://www.softimage.com/products/behavior.

[SWND03] SHREINER D., WOO M., NEIDER J., DAVIS T.: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4*. Addison-Wesley, 2003.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2 (March-April 2002), 36–43.

[Tob03] TOBITA H.: VelvetPath - layout design system with sketch and paint manipulations. In *Proc. Eurographics '03 Short Presentations* (2003).

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation. *Computer Graphics Forum 21*, 4 (Dec. 2002), 767–775.

[Wol02] WOLFRAM S.: *A New Kind of Science*. Wolfram Media, Inc., 2002.

[WS02] WAND M., STRASSER W.: Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum 21*, 3 (2002). (Proc. Eurographics'02).

[XLTP03] XU S., LAU F. C. M., TANG F., PAN Y.: Advanced design for a realistic virtual brush. *Computer Graphics Forum 22*, 3 (2003), 533–542. (Proc. Eurographics'03).

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: An interface for sketching 3D scenes. In *Proc. SIGGRAPH '96* (1996), pp. 163–170.