EG UK Theory and Practice of Computer Graphics (2007)
Ik Soo Lim, David Duce (Editors)

The Dynamic Animation of Ambulatory Arthropods

L. ap Cenydd[†] and W. Teahan[‡]

School of Computer Science, University of Wales Bangor, UK

Abstract

Whilst advances in real-time computer graphics continue to permit the development of increasingly vivid virtual worlds, the degree of interaction between the environment and the animated characters within remains relatively limited. There has been little research into the realistic real-time simulation of creatures with the ability to scale arbitrary surfaces and fully explore their environment. Natural looking animations of such feats would greatly enhance immersion in computer games, as well as being of benefit to fields such as phobia therapy and Artificial Life research.
We present a system for dynamically animating ground based arthropods in real-time, capable of traversing realistically across a complex, arbitrary environment. The physical simulation of the virtual world further grounds the creatures, enabling complex emergent animations to form.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism-Animation; I.6.8 [Simulation and Modeling]: Types of Simulation-Animation

1. Introduction

Most insects and arachnids are capable of unrestricted movement within their environment. They can navigate cluttered surfaces, scale walls and even walk across ceilings with relative ease. Despite their evolutionary simplicity, they display a vast range of dexterous motions in getting from one surface to another, the subtleties and variance of which are impossible to capture using key-frame animation and its commonly associated techniques.

We have developed a simulation for dynamically animating the majority of ground based arthropods in real-time, capable of displaying the range of motion and dexterity (and somewhat inherent lack of grace) found in real-life equivalents. By physically simulating the creatures and the virtual world around them, we achieve a level of agent-environment interaction simply not possible using more abstract representations. By *grounding* the simulated creatures in this way, more natural animation is generated.

Our implementation is limited to simulating arthropods with six or eight legs only, and excludes insects capable of flight.

The rest of the paper is as follows. Section 2 contains an overview of previous and related work, section 3 details the physical creature model and associated graphical fidelity, section 4 explains the Gait Generator, section 5 the Behavioral System, and finally section 6 the Virtual Environment.

2. Related Work

The dynamic animation of articulated characters is a far from trivial problem, and there has been much research concerning legged locomotion synthesis and procedural gait generation. This section provides a brief overview of related and previous work.

2.1. Gait Generation

A substantial survey of work related to both physically and kinematics based human gait generation can be found in [MFCD99]. Issues discussed of relevance to our work includes arbitrary path gait adjustment, and step adjustment across uneven terrain. Sun et al. [SM01] addressed these two issues using three modular components, allowing for curved

[†] Email : llyr@informatics.bangor.ac.uk
[‡] Email : wjt@informatics.bangor.ac.uk

path and uneven terrain locomotion using sagittal elevation angles and an inverse motion interpolation algorithm. An alternative technique can be found in [kCH99], where the interactive animation of human figure locomotion is generated by a hierarchical motion control system, with pre-processed motion modules and controlling parameters applied to alter walking style and personality.

Regarding arthropod gait generation, much research has been undertaken in the fields of biology [Bow81] and robotics [Fer95] [BQCR97], the relevant parts to our work being the hexapod and octopod base walking gaits, and the dynamic adjustments in said gait due to changes in the creature's speed, heading and environment.

In [MZ90], the coordination of a kinematic cockroach was automatically generated by a gait controller, with stance and motor programs applying appropriate forces to the legs. Parker [Par98] used cyclic genetic algorithms to evolve a variety of arachnid walking gaits, which were tested on a physical based model of an eight legged robot.

Cruse et al. [CDK*99] assumed that a decentralized control scheme for each of their simulated insect's limbs led to a flexible system, using artificial neural networks.

Klaassen et al. [KLSK01] developed a biomimetic control scheme for an octopod robot, their approach based on two biological control principles, the Central Pattern Generator (CPG) and the Reflex. The two principles are combined with Basic Motion Patterns (BMPs), which describe each leg's trajectory during a stride. Under normal circumstances the robot's legs are controlled directly by the CPG, with the Reflexes only activated during exceptions, such as an object blocking the leg's path.

2.2. Previous Work

In our previous implementation [aCT05], some of the techniques above relating to gait and stepping functions were combined with point a mass behavioral system based on Reynolds' *Steering Behaviors* [Rey99], the navigation of the environment surface governed by an orientation system similar to [Ste97]. This approach was used in [Sul06] to create a pre-rendered animation featuring a large number of procedurally generated spiders.

As far as we are aware, there has been no other research into real-time simulation of ambulatory arthropods in an arbitrary environment, let alone the issues associated with climbing and scaling large changes in surface orientation.

3. An Arthropod Model

Our simulation's creature model consists of a low polygon mesh rigged with a pseudo-anatomically correct skeleton.

The low-poly mesh serves as the visual model for the default foreground representation of the creature. Creature models can also be constructed from sub-meshes, allowing the construction of a multitude of creatures that are close enough in visual appearance (for example the same subphylum) so that their limbs are interchangeable. As the simulation allows for each creature and object in the environment to be scaled independently, the actual scale of the mesh is largely irrelevant, greatly facilitating the development of content for the simulation.

An extremely high-poly mesh is constructed by subdividing and adding definition and detail. The data from the model is then captured in normal, specular and displacement maps that can be applied to the low-poly mesh at runtime, retaining much of the high-poly detail and futher enhancing the appearance of the simulated creature.

3.1. Rendering Fur

Filamentous outgrowths are a distinctive characteristic of several arthropod species, resembling sparse strands of thick hair, particularly common along the creature's jointed appendages. Furthermore certain types of Spider, especially of the Tarantula species, are covered in a thick coat of finer hair.

Our simulation is capable of automatically rendering both types of hair, either singularly or together, given a mesh and corresponding texture maps. Moreover, the technique can be extensively tweaked if greater control over the placement and attributes of the hair fibres is required.

Our rendering technique is based on *fins* and *shells* [LPFH01], where the former method is used for generating the sparse, fibrous hair and the latter, or a combination of both, when generating thicker fur. Example uses of both techniques can be seen in Figure 1.

3.2. Shells

The basic premise behind the shells technique is to render a series of layers, or shells, at increasing distances from the model, thereby mimicking a minimized textel volume around the surface.

Our implementation uses a procedural noise algorithm to generate a four channel (RGBA) texture for each shell with parameters for adjusting different fur properties (sparse, thick, kinked etc) at each iteration. By default at each consecutive layer the alpha channel brightness (and therefore the opacity of the shell) is reduced to simulate tapering of the fibers. Similarly, the RGB brightness is increased at each iteration, facilitating an illusion of depth.

The main drawback of this technique is that visual quality is heavily dependant on the number of shell iterations. However, as our requirements are based on arthropod hair simulation, an adequate result can be achieved with 8 concentric shells, avoiding the more computationally expensive requirements of generating longer hair fibre effects.

Top image : Fins only
Bottom image : Both fins and shells

Figure 1: *Tarantula model with fur shaders applied*

3.3. Fins

The fins technique involves the creation of textured polygon strips (fins) along the edges of the base model, the fin extruded along the normal of each edge.

For both efficiency and aesthetic purposes, the opacity of the fins are adjusted relative to the viewer. The fins are fully opaque when perpendicular to the camera (and therefore visible around the silhouette of the model), and fade to eventual transparency as the viewing angle tends towards parallel. This fin fading works best in conjunction with shells, as whilst the former works well around the model silhouette, the latter offers the best visual approximation when the surface is orientated toward the camera.

4. The Skeleton

In order to generate a realistic simulation of an arthropod, it is essential that the underlying skeleton is as accurate and anthropomorphically correct as possible, whilst simultaneously avoiding a degree of complexity that would result in a negligible increase in animation accuracy for invested computational power.

4.1. Skeletal Hierarchy

The structural information for a creature's skeleton is stored in a *.skeleton* XML file, which contains a list of all bones in the skeleton, with each bone containing an integer id number, name string, its local position vector, orientation axis vector and offset rotation angle. Information pertaining to the skeletal hierarchy is also stored.

Naming conventions for individual skeletons are fairly uniform to better facilitate multiple individual species using the same underlying sub-skeleton data. Unless an accurate as possible simulation is required, it is useful to merge less supple joints into more dominating ones in order to simplify inverse kinematic solver calculations, the trade-off being a small loss of dexterity in the simulated appendage.

Figure 8 shows an underlying Huntsman Spider skeleton. The final joint in the limb hierarchy, being the end-effector of the leg bone chain, has no direct effective influence on the skeleton itself (and therefore the mesh skin). Its purpose is to serve as an end to the local hierarchical chain, and denotes the position of the foot on each appendage.

Data relating to the specific simulation attributes of the bones (limits, springs, degrees of freedom etc.) are not stored in the skeleton file, the reason being the previously discussed ability to interchange limb meshes. Separating the skeleton structural data from the organic creature attribute data relating directly to the creature's eventual dynamic animation allows for multiple alternative configurations for the same underlying skeleton (and corresponding skin mesh). The skeleton data file only contains information on the relative position of the bones and their default orientation within the skeletal structure.

4.2. Relative Skeletal Inference

The simulation also requires each bone in the hierarchy to be set to its default (comfortable) position. That is, when the creature is initialized within the simulation on a flat surface, it will retain the exact pose described in the skeleton file, unless overruled elsewhere. This provides a default stance for each individual skeleton, which eliminates the need for every creature that shares the same skeletal limb data to have its own list of joints and corresponding rest orientations and rotational axis. Secondly it aids the inverse kinematics solver module, providing a reference idle configuration by which all joint rotations are measured.

By default, all attributes relating to joint restrictions, limitations and dexterity are calculated relative to the initial skeletal configuration. For example, if a skeleton femur joint is initially orientated at a 45° angle along its local x axis

within the bone hierarchy, and it is specified within a creature's attribute data that the femur joint has a limitation of +25° and -11°, the leg will have an absolute limit of rotation at 70° (max) and 34° (min), allowing for freedom of movement over 36° overall.

5. The Gait Generator

Figure 2 shows a diagram of the simulation's components. At the core of the locomotion layer is the Gait Generator, which is responsible for directing and maintaining a stable, cohesive and natural looking walking animation. Its main function is to direct when, where and how each of the creature's legs move, via a Stride Module. It also oversees all of the regulatory modules, which track the creature's functions, and act on any flags raised by said modules.

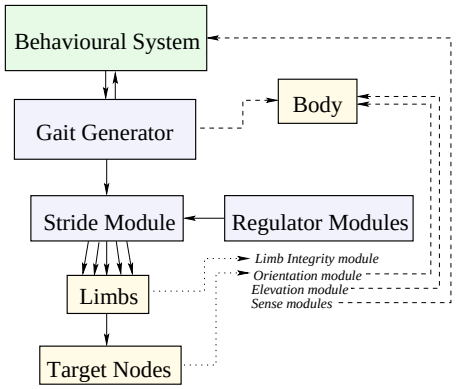


Figure 2: *Component relationship diagram*

This section provides an overview of the major components of the Gait Generator. Due to limited space, implementation specifics can be found in [aCT07].

5.1. Creature Attributes

As previously mentioned, each creature has a corresponding attribute file which provides species specific information on dexterity and locomotion characteristics. With joint axis, limits and sensitivity being previously passed to the creature's limbs (and corresponding inverse kinematics solver) , the rest of the attribute data is used by the Gait Generator to setup and procedurally animate the creature, and includes both global and local (limb specific) information . Below is a list of some of the most important parameters.

- *MaxSpeed (global)* - maximum speed of creature;
- *initElev (global)* - default elevation of the creature;
- *initDuration (global)* - default duration of a single stride;
- *initOrientSens (global)* - sensitivity to orientation changes;
- *initOrientBias (global)* - frontal orientation bias;
- *MaxStride (local)* - maximum distance of a stride;
- *MaxHeight (local)* - maximum height of a stride.

5.2. Taking a Step

The creature's absolute position and orientation is governed by a rigid body, with a convex hull approximating the mesh collision structure. With each stride triggered, a virtual push force is applied to the rigid body, pushing the creature in the eventual direction of the step. The size of this force is based on a global target velocity, requested of the Gait Generator by the higher level Behavioral System.

Whilst the creature is in uninhibited locomotion, the Gait Generator instructs each leg to take a step in sequence. The order by which the legs step differ between arthropod subphyla, which our simulation simplifies by making a distinction between hexapod and octopod gaits.

Each limb has a corresponding target node, independent of the creature's position. The target node is a fully simulated spherical rigid body which can be seen as the virtual position of the limb's extremity, or the ideal position of the end-effector at the current simulation step. By incrementally updating each limb's joint chain, it is the goal of the inverse kinematics solver to make the transition from one target node position to the next as effortless and natural as possible.

The Stride Module uses a procedural step generator to define the path each target node takes when the associated leg is in its swing phase. The source for this stride path ranges from simple ellipsoid functions to advanced Bézier curve representations, depending on the type of step being performed. Furthermore, the baseline path can be transformed or replaced mid-stride by the Gait Generator in the advent of some internal factor (such as a behavioral change) or external factor, the foot failing to strike a surface for instance.

Although more computationally expensive, Bézier curve representations allow for more elaborate, asymmetrical stride paths more commonly seen in nature. It is also possible to create modular strides, where several Bézier curves are spliced together at certain points along the path.

The specifics of deciding what type of curve representation to use for each stride, along with any extra conditions such as previously mentioned multiple Bézier curve compositions, can be found in [aCT07].

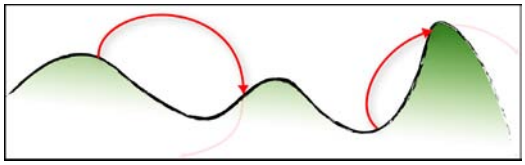
An example of a simple path representation can be seen in figure 3. In the absence of any external obstacle, the ellipsoid path shown would simply result in the leg probing forward before retracting back to its original position.

5.3. Stride Variables

The speed by which the stride is taken is governed by a step duration, a variable which is linked with the current speed

of the creature, and passed to the Stride Module by the Gait Generator. The division of the time elapsed in the current stride with the step duration gives the percentage of the stride currently complete. Therefore using the example path shown in figure 3, at 25% complete the target node (and therefore the limb end-effector) would be at its highest vertical point, at 50% its maximum horizontal distance from origin, with beyond 50% representing the limb's retracting phase.

In order to translate the 2D path cartesian coordinates into actual target node movements in simulation space, the paths are translated and rotated by the appropriate magnitude and orientation. The target node itself is updated by changes in velocity, the acceleration being the change in tangent along the path from one time step to the next. Given the unit y component of the quaternion representing the creature's orientation, and $vDirection$, the proposed vector direction of the step, the velocity of the target node for the next frame can be calculated by: $velocity = ({}_vDirection * coord.x) + ({}_vUnitY * coord.y)$, where $coord$ is the current path coordinates, scaled by the width and height of the current step.



The red paths show two example steps, and the point in which they will intersect the environment and terminate the stride.

Figure 3: *Path trace on uneven surface*

5.4. Anchoring on the surface

The termination of a stride usually occurs when the target node (or end-effector) collides with the surface, or an object of significant mass. Possible collisions between the target nodes and the environmental mesh(es) are tested every frame, up until such an event occurs and the target anchors in place. The end-effector will remain in that position until the next stride is triggered, becoming a pivot for the rest of the creature. Such functionality allows for the creature to take blind strides, only terminating when the leg strikes something which impedes its progress further. When walking across a completely flat surface this equates to each leg's stride path anchoring when the path trace is 50% complete.

5.5. Landscape Orientation

In our previous implementation, the orientation of the creature's body relative to the local surface was set by a simple ray-polygon intersection test, the ray cast from the origin of the creature's body in the direction negative to it's current up-vector. This corresponded to an observation made by the primary author that the orientation of an arthropod was always roughly parallel to the surface currently being traversed. Creating an edge connectivity map of the environment mesh(es) at initialization allowed for fast access to the nearest polygon's normal, and therefore the target orientation of the creature. The intersection test was carried out every set number of simulation steps, where the orientation transition was interpolated, the interpolation rate being relative to the change in orientation angle.

The inherent problem with this technique was that whilst adequate for small changes in orientation over suitably convex or concave surfaces, environments consisting of irregular, chaotic or largely angular terrain were difficult to traverse, as the orientation changes became frequent and erratic. Furthermore, when encountering changes in surface angle of 90° and beyond, the ray-intersection test fails. Our first attempt at solving this problem, by scanning the orientation ray at increasingly larger angles in an attempt to locate an orthogonal polygon, whilst proving a viable solution for relatively simple environments (such as a cube or other primitive), did not produce satisfactory results for arbitrary complex environments.

5.6. Touching the Surface

Our current implementation approaches the orientation of the creature from a purely physical, grounded perspective.

The absolute orientation of the creature is governed by the Orientation Module, the aim of which is to orientate the creature's body relative to the positions of each of its anchored feet. Whilst in our previous implementation the alignment of the creature was based on terrain-ray intersection, this approach considers the internal configuration of the skeleton itself, and therefore the orientation of the creature is directly influenced by the anchored position of each leg on the landscape. As the inverse kinematics solver continually maintains contact between the end-effectors and their corresponding target nodes, the feet remain anchored in position as the legs flex and the creature's body re-orientates. As there is constant feedback from the feet as they anchor the surface at the termination of each stride, the orientation of the creature is continuously updated, maintaining an accurate stance configuration at all times.

The Orientation Module is updated each time a foot anchors on the surface, with the leg completing the swing phase of its cycle and entering the stance phase. The module can also be updated during the swing phase, thereby taking the foot's position whilst in transit into account, resulting in erratic micro changes in orientation during locomotion which can be useful in creating a more accurate animation of certain species of arthropod, especially longer limbed Arachnids where leg movements are more pronounced. The module compares the elevation between relevant limb target nodes at each update, calculating two components – the pitch and the roll. The yaw component is not calculated, as it is taken directly from the creature's steering vector.

The pitch component is found by calculating the normalized average vector between adjacent target node positions along the creature's *x* axis, parallel to its heading. The relative positions of the target nodes along the *z* axis are ignored, as their influence is calculated in the roll component. The pitch is therefore the difference in elevation between frontal and backward limb couplings. Similarly, the roll component is found by normalizing the average vector between leg pair target nodes perpendicular to the creature's heading, along the *z* axis. Here, the *x* axis position is ignored, as its influence is included in the pitch component, the resulting vector being the elevation of one side of the body as compared to the other.

The calculations for both these components can be simplified by only taking the difference in elevation between front and back limb couplings into account, with more accurate calculations only requested when the creature detects a large change in elevation, either because its frontal limbs have anchored at a sufficiently different elevation or a flag is raised in its sense modules.

In order to update the creature's rotations along the three separate axis the current quaternion describing the orientation of the creature is multiplied by the corresponding unit vector for each pitch, yaw and roll component. The torque of each component is calculated by: $Torque = (axis \times \Delta c_1) - (\Omega c_2)$ where: *axis* is the rotational axis, Δ is the angle between current and target component vectors, Ω is the current rotational velocity of the creature and the c_1, c_2 variables represent the rotational rate and decelerations respectively.

Varying c_1 allows for faster or slower reactions to orientation changes, which can not only be changed to reflect different gaits, locomotion speeds and underlying terrain structure, but also the global sensitivity to changes in orientation – allowing for more jittery or relaxed reactions to surface changes. For example, setting c_1 to a higher value for a Domestic House Spider (Tegenaria domestica) would reflect its smaller, more erratic nature compared to a larger arachnid like the Tarantula, where a lower value would reflect its slower, more deliberate gait.

When calculating the pitch component of orientation, bias can also be applied to the front pair or quadrant of limbs. This amplifies the creature's sensitivity to changes in frontal elevation, which is useful when traversing massive changes in surface angle. As the frontal limbs are usually the first to come into contact with the feature, this frontal bias allows the creature to react faster and more realistically to the surface. As with the reaction time variable c_1, the bias can be altered during simulation to reflect the characteristics of the underlying terrain.

5.7. Grounding the creature

Regulating the creature's orientation in this way has several distinct advantages compared to our previous ray intersection implementation. Not only does it maintain a realistic and comfortable looking orientation for the creature with respect to its limb configuration, it also greatly enhances the locomotion system as a whole, as consecutive strides of trailing limbs are affected by previous strides due to the constant feedback the creature receives from the environment.

A typical scenario that highlights this effect is a creature approaching a precipice from a relatively flat surface. The creature maintains its standard walking gait and stable orientation until one if its front two legs goes over the edge and anchors in the surface at a lower elevation relative to its body. This causes the orientation of the creature to tip forward. The creature continues to step with its other legs, only now the paths of the strides are directed along this new heading. When the other frontal foot anchors on the cliff, it tilts the creature further over the edge, and so on. The effect is that past strides influence the direction of present ones, guiding the orientation and therefore the velocity of the creature naturally and realistically over the edge and eventually onto the new surface (see Figure 6).

6. Behavioral System

The Behavioral System contains the bulk of the creature's AI, and is responsible for directing where the creature moves in the environment (see [aCT05], [Rey99]), as well as making higher-level decisions regarding locomotive issues. The subsumption of the Gait Generator's lower-level hardwired locomotion scheme by the latter function is based on [KLSK01]. Examples of such reflexes are *Stance* and *Search*.

6.1. Stance Reflex

If the creature changes its heading whilst idle, or the Limb Integrity Module detects that a leg is significantly stretched, the Behavioral System will trigger a Stance Reflex, which overrides the Gait Generator and requests that the Stride Module step the leg back into a more comfortable position.

6.2. Search Reflex

The Search Reflex is triggered when a foot fails to strike the surface at the end of the leg's swing phase, an example being a creature walking up to a precipice at significant speed. This reflex aims to make contact with the surface by randomly probing the leg in increasingly concentric circles. This reflex is based on insect stereotypic leg searching movements [Dür01].

7. Interacting with the Environment

Our simulated creatures are designed to fully explore an arbitrary environment, including environments containing objects of differing materials. Every sub-mesh that exists in the

world has a corresponding collision mesh, allowing physical contact between creature and object (see Figure 9). Furthermore, objects are rigid bodies with mass, size and shape, and can therefore interact with other objects in the scene.

7.1. Material Pairs

Every rigid body has an associated material which describes its properties. When two bodies collide, the materials are accessed and the event processed. The pair of materials can have an associated collision response, which describes what happens under such a condition. Whilst purely aesthetical effects are possible, such as an appropriate amount of dust being kicked up from a sandy material (magnitude being linked to velocity), the true power of this functionality comes in further aiding the creature's grounding in the environment, enhancing the resultant behavioral animation in the process.

Material pairs allow a greater degree of tactile feedback between creature and environment, impossible using collision meshes only. Every surface can be given friction properties that only come into effect when the creature's foot (or target node) collides with it. A glass material, for example, would have corresponding low friction, resulting in the creature failing to anchor its foot if the collision angle was above a certain threshold. The effect of this would cause the creature to slide its leg down the glass and eventually enter the stride's retraction phase. Eventually, the impeded progress would force the creature to alter its approach.

Another purely physical example is that of a creature traversing a forest floor environment, cluttered with detritus (rocks, bark, leaves etc). If the creature had significant mass, the act of walking on an object might disturb it. By processing the event, the simulation is able to determine the effects and whether the creature can maintain its foothold.

8. Conclusion

Our simulation is capable of procedurally generating locomotive animations for a diverse range of arachnid and insect species, the realism evaluated by comparing with footage of real-life equivalents. The creatures are capable of traversing a wide range of environments, the exception being surface changes approaching 180° and terrain with irregular footing. We are currently developing the Behavioral System and implementing additional reflexes for these circumstances, as well as taking further advantage of object material properties.

Example screenshots can be seen in Figures 4 - 9.

References

[aCT05] AP CENYDD L., TEAHAN W.: Arachnid simulation : Scaling arbitrary surfaces. In *EG UK Theory and Practice of Computer Graphics* (2005).

[aCT07] AP CENYDD L., TEAHAN W.: *Walking Arthropod Simulation.* Tech. rep., Uni. of Wales, Bangor, 2007. http://www.informatics.bangor.ac.uk/~llyr/.

[Bow81] BOWERMAN R.: Arachnid locomotion. *Locomotion and Energetics in Arthropods* (1981), 73–102.

[BQCR97] BEER R. D., QUINN R. D., CHIEL H. J., RITZMANN R. E.: Biologically inspired approaches to robotics: what can we learn from insects? *Commun. ACM 40*, 3 (1997), 30–38.

[CDK*99] CRUSE H., DEAN J., KINDERMANN T., SCHMITZ J., SCHUMM M.: Walknet - a decentralized architecture for the control of walking behavior based on insect studies. *Hybrid Information Processing in Adaptive Autonomous Vehicles* (1999).

[Dür01] DÜRR V.: Stereotypic leg searching movements in the stick insect: kinematic analysis, behavioural context and simulation. *J. of Experimental Biology 204*, 4 (2001).

[Fer95] FERRELL C.: A comparison of three insect-inspired locomotion controllers. *Robotics and Autonomous Systems 16* (1995), 135–159.

[kCH99] KAI CHUNG S., HAHN J. K.: Animation of human walking in virtual environments. In *CA* (1999), pp. 4–15.

[KLSK01] KLAASSEN B., LINNEMANN R., SPENNEBERG D., KIRCHNER F.: Biomimetic walking robot scorpion: Control and modelling. In *Proc. 9th Int. Symposium on Intelligent Robotic Systems* (2001), pp. 101–108.

[LPFH01] LENGYEL J. E., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. In *Symposium on Interactive 3D Graphics* (2001), pp. 227–232.

[MFCD99] MULTON F., FRANCE L., CANI M.-P., DEBUNNE G.: Computer animation of human walking: a survey. *J. of Vis. and Comp. Animation 10* (1999), 39–54.

[MZ90] MCKENNA M., ZELTZER D.: Dynamic simulation of autonomous legged locomotion. *Computer Graphics 24*, 4 (August 1990).

[Par98] PARKER G.: Generating arachnid robot gaits with cyclic genetic algorithms. In *Genetic Programming 1998: Proc. 3rd Ann. Conference* (1998), pp. 576–583.

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters. In *Game Developers Conference 1999* (1999).

[SM01] SUN H. C., METAXAS D. N.: Automating gait generation. In *SIGGRAPH* (2001), pp. 261–270.

[Ste97] STEED A.: Efficient navigation around complex virtual environments. In *VRST '97: Proc. of the ACM symp. on Virtual reality software and technology* (1997), ACM Press, pp. 173–180.

[Sul06] SULLIVAN C.: *Technical Animation Using Maya and Mental Ray*. Master's thesis, Bournemouth Uni., 2006.

Figure 4: *Huntsman spider idle on rock.*

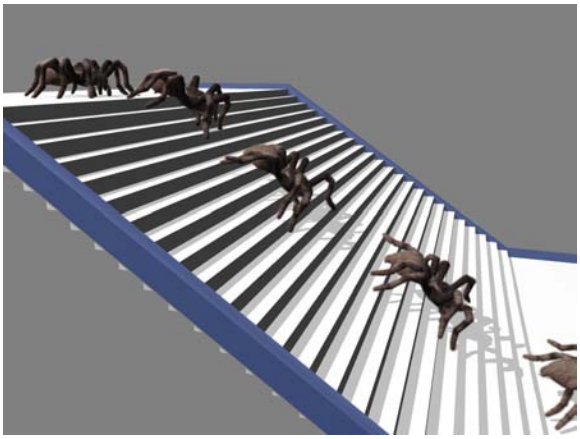


Figure 5: *Sequence of large Tarantula traversing stairs. Creature scale 10.0.*

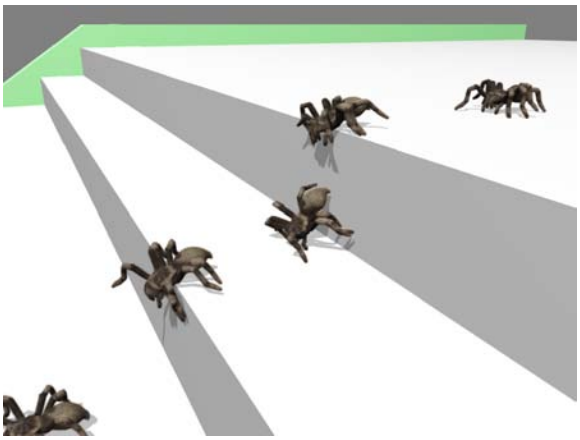


Figure 6: *Sequence of small Tarantula traversing stairs. Creature scale 1.0.*

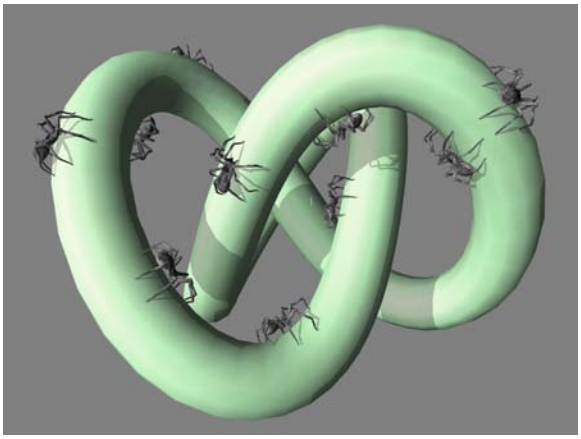


Figure 7: *Select frames of two Huntsman spiders walking on a Torus Knot.*

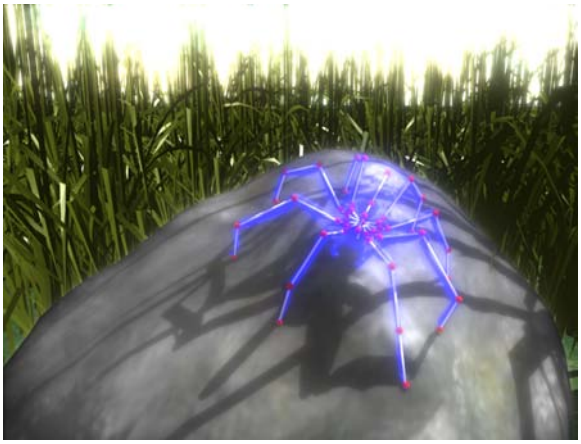


Figure 8: *Underlying Huntsman spider skeleton.*

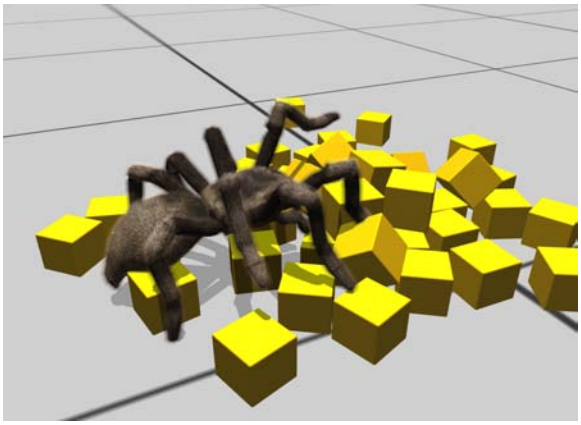


Figure 9: *Tarantula climbing over a pile of rigid bodies.*