KinÊtre: Animating the World with the Human Body

Jiawen Chen, **Shahram Izadi**, **Andrew Fitzgibbon**
Microsoft Research Cambridge, UK
{jiawen, shahrami, awf}@microsoft.com

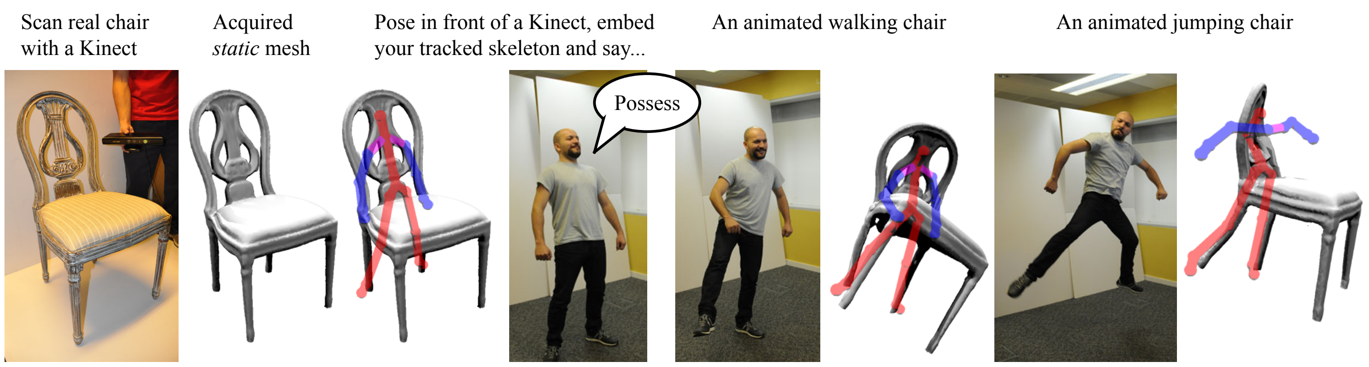
Scan real chair with a Kinect
Acquired *static* mesh
Pose in front of a Kinect, embed your tracked skeleton and say...
Possess
An animated walking chair
An animated jumping chair

Figure 1: KinÊtre allows novice users to easily and quickly create animations of arbitrary real-world objects using just a regular consumer Kinect. From left to right: user scans a real chair using an existing 3D reconstruction system. The Kinect is then placed down so that it can image the user's full body, and a real-time tracked human skeleton is acquired. Our system takes the human skeleton and 3D reconstruction as input. Next, the user physically moves such that the rendered skeleton interpenetrates the mesh. Using a simple voice command, the mesh is rigged (or "possessed") in real-time by the user's skeleton. KinÊtre transfers motions of the user into realistic deformations of the mesh.

ABSTRACT

KinÊtre allows novice users to scan arbitrary physical objects and bring them to life in seconds. The fully interactive system allows diverse static meshes to be animated using the entire human body. Traditionally, the process of mesh animation is laborious and requires domain expertise, with rigging specified manually by an artist when designing the character. KinÊtre makes creating animations a more playful activity, conducted by novice users interactively *at runtime*. This paper describes the KinÊtre system in full, highlighting key technical contributions and demonstrating many examples of users animating meshes of varying shapes and sizes. These include non-humanoid meshes and incomplete surfaces produced by 3D scanning – two challenging scenarios for existing mesh animation systems. Rather than targeting professional CG animators, KinÊtre is intended to bring mesh animation to a new audience of novice users. We demonstrate potential uses of our system for interactive storytelling and new forms of physical gaming.

Author Keywords

3D interfaces; mesh animation; real-time; depth cameras

ACM Classification Keywords

H.5.2 [Information Interfaces And Presentation]: User Interfaces - Interaction styles

INTRODUCTION

Imagine you are asked to produce a 3D animation of a demonic armchair terrorizing an innocent desk lamp. You may think about model rigging, skeleton deformation, and keyframing. Depending on your experience, you might imagine hours to days at the controls of Maya[1] or Blender[2]. But even if you have absolutely no computer graphics experience, it can be so much easier: grab a nearby chair and desk lamp, scan them using a consumer depth camera, and use the same camera to track your body, aligning your virtual limbs to the chair's geometry. At one spoken command, your limbs are attached to the chair model, which follows your movements in an intuitive and natural manner.

KinÊtre[3] is such a system. Rather than targeting professional animators, it brings animation to a new audience of users with little or no CG experience. It allows realistic deformations of arbitrary static meshes, runs in real time on consumer hardware, and uses the human body for input in conjunction with simple voice commands. KinÊtre lets anyone create playful 3D animations.

To realize KinÊtre, we extend the work of Sumner et al. [32] to handle incomplete 3D meshes captured by a real-time 3D reconstruction system [15]. Rather than using mouse or touch, our technique leverages the human skeleton tracked by a Kinect [30] as a more direct and higher degree-of-freedom (DoF) mechanism for mesh animation.

We describe our system pipeline in full, and demonstrate many examples including animation of incomplete physical scans of varying size and shape, retargeting motion to other human and non-human meshes, and animations with multi-

[1]http://usa.autodesk.com/maya/
[2]http://www.blender.org/
[3]A portmanteau of Kinect and Être—"to be."

user input. We describe usage scenarios enabled by this combination of capturing arbitrary real-world objects and bringing them to life using the human body: for example, animating characters for home scripted CG movies or interactive storytelling – imagine enacting the classic Pixar Luxo Jr. scene, by first scanning a household lamp and then controlling it using your body. Other scenarios include creating more realistic avatars for games, where users scan themselves or friends and again control these meshes using their bodies, without any manual rigging or skinning process, as well as leveraging these types of playful mesh animations for physics-enabled gaming. We conclude by discussing qualitative experiences of using the system, its strengths and limitations, and directions for future work.

Our main contributions include:

- A new interactive mesh animation system that allows non-expert users to rapidly create playful animations, using scanned physical objects or existing 3D models.
- A new deformation method, based on [32], capable of performing realistic and detail preserving deformations of incomplete static input scans using the motion of a tracked user's skeleton as real-time input.
- A demonstration of our results operating on input meshes of different topology, size and geometry, and enabling new scenarios for character animation and physics-based gaming previously unavailable to everyday consumers.

RELATED WORK

While considerable advances in real-time virtual character animation have occurred over many decades (see [31] for an overview of pioneering work), this process is still largely based on specialist tools designed for professionals to create high-end animations for movies and games. Our goal is greatly different, we instead target novice users with little or no background in CG tools. Our primary goal is to allow users to rapidly incorporate objects from the real world into their CG applications, and bring these static representations to life in seconds. We leverage the human body for input, instead of mouse and keyboard, to create a more direct performance-based approach to animation, making the process more playful and accessible to a wide range of users.

Manual skinning and skeletal-based deformations remain the de facto method for character animation in a variety of CG applications including gaming. For example, when we see virtual characters coming to life in our games, usually a designer has carefully constructed and *rigged* the mesh, involving embedding a skeleton within the target model and painting individual bone weights onto each vertex. At runtime, motion capture (mocap) data or an inverse kinematics engine drives the character's bones, which then transforms the mesh.

The 3D modeling, rigging, and animation process traditionally requires a great deal of time and expertise and has led to extensive research in automatic animation systems. These include sketch-based interfaces for 2D and 3D animation [33, 9], and the use of multi-touch input for both manipulation of unstructured 2D shapes [13] and 3D articulated human characters [21]. More unusual interfaces include using physical manipulators such as soft toys [18], articulated robots [22], 2D paper cutouts [5], and layering multiple passes of 3D mocap [10]. Mesh manipulation tools can be spatially combined with key-framing to assist artists in scripting animation [14]). Our focus is on real-time 3D shape animation, leveraging the natural and higher DoF input of the entire human body for control.

Another related area explores generating articulated meshes of humans using multi-view video [1] or silhouettes [34], range data [2] and mocap data [3]. These approaches simultaneously acquire and articulate human meshes, but clearly cannot scale to scenarios where animations of non-humanoid characters are desired.

Closer to our goals are systems that leverage recorded or live mocap data to animate meshes [4] or articulated characters [29]. These systems demonstrate compelling examples of character animation on various closed meshes, predominately of humanoid forms. Our method is more flexible in the types of 3D deformations and meshes it supports, including incomplete scans of arbitrary physical objects. Our focus is on a "user in the loop" form of interaction, rather than the more sophisticated technique demonstrated in [4], which works particularly well for closed humanoid meshes but is restricted to watertight shapes. Recent work by Yamane et al. [35] takes a learning-based approach to mapping human motion to non-humanoid characters. Given user-selected correspondences between the poses of the human actor and those of the non-humanoid character, the system builds a statistical mapping; which, combined with physical constraints, transfers the actor's motion to the character. In contrast, KinÊtre is fully automatic and does not require users to pose the character, only themselves. However, the motions that KinÊtre produces are "direct" mappings and not as flexible as those of Yamane et al.

Recently, there has been much work in the area of shape-preserving space deformation algorithms. One approach is to operate on a reduced control *cage* surrounding the mesh, which can have their handles (vertices of the cage) manipulated to cause smooth, detail preserving surface deformations. These cage-based methods typically restrict cage vertex transformations to a series of translations, and use generalized coordinate techniques to uniquely define any possible configuration of the mesh relative to a reference configuration [20, 25, 6, 19]. Another class of techniques support manipulation through unconnected point handles on the mesh. Examples include moving least squares [27], variational methods [6], and inverse kinematics on meshes [28]. These systems all operate on closed meshes, which cannot be guaranteed when scanning real-world geometry. Further, they are optimized to support manipulation of control vertices by mouse or stylus.

The 3D deformation methods described so far all operate on closed meshes. Jacobson et al. [16] demonstrate bounded biharmonic blending weights that produce smooth and intuitive deformations for points, bones and cages of arbitrary topology including incomplete meshes. However, it relies on a volumetric representation and quadratic programming, which incurs a high computational cost each time the user changes the set of constraints. Embedded Deformation by

Sumner et al. [32] supports point-based direct manipulation of meshes, and is agnostic to whether surface data is complete. This method forms the basis of the deformation model used within this paper. We extend embedded deformation in various ways to our novel scenario of use, where data is heavily unstructured and multiple nodes of the graph are manipulated simultaneously by a human skeleton.

SCENARIO OF USE

Before going into the details of our system, we begin with a motivating scenario (shown in Figure 1), to help illustrate the overall user experience and interaction flow.

Imagine that the user wants to introduce and animate an existing static physical object – a chair – into a 3D game. He picks up a Kinect camera and scans in part of the room. KinÊtre automatically generates a 3D mesh of the chair and segments it from the background (Figure 1 left).

At this point we have a static, incomplete and un-rigged mesh in our CG world. To bring this object to life in realistic ways, the user first places the depth sensor back down so that it can image his full body, and a tracked human skeleton is sensed. Our system renders both the human skeleton and scanned object side by side. The user then physically moves such that the desired parts of the rendered skeleton are embedded inside the scanned object (Figure 1 middle). Once in the desired pose, the user issues a voice command to automatically rig (or "possess") the mesh (Figure 1 middle). This attaches joints of his current pose to the mesh.

The user can now move his body and KinÊtre automatically translates his motion into realistic deformations of the previously static mesh (Figure 1 right). Moving his legs causes the legs of the chair to also move. Flexing his knees causes the chair to bend, jumping up makes the chair also jump. Motions of the body propagate in real-time to the mesh and provides a natural way for users to perform playful animations of any object that they wish to bring to life in their CG application or game.

SYSTEM PIPELINE

Although our system works with arbitrary meshes, including complete high-quality 3D models, typically, the input to our system is a real-world object of reasonable physical size and surface reflectance. We acquire the object's 3D geometry by waving a depth camera around it, using a variant of the KinectFusion system [15].

The main pipeline shown in Figure 2 is comprised of two main phases. The first phase can be thought of as *preprocessing* and is broken down as follows:

1. A *3D reconstruction* step estimates the 6-DoF pose of the moving Kinect camera, and fuses depth data continuously into a regular 3D voxel grid data structure.

2. A *segmentation* and *meshing* step automatically extracts and triangulates the desired foreground isosurface stored implicitly in the voxel grid.

3. A *sampling* step traverses vertices of the mesh and constructs a sparse deformation graph [32], with the goal of maintaining key topological features and connectivity of the mesh. This deformation graph is the basis for representing deformations in the higher dimensional mesh.

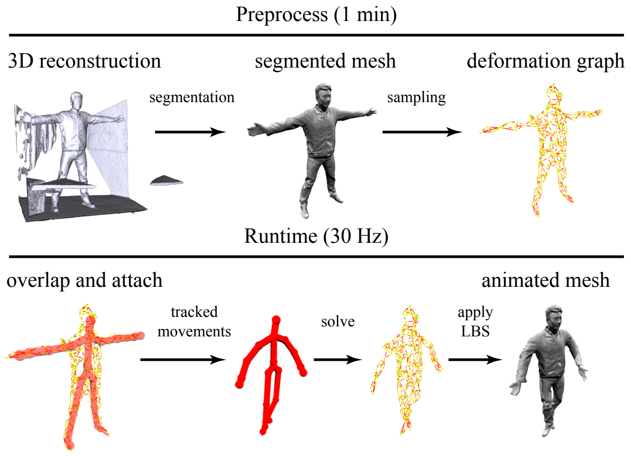

Figure 2: The KinÊtre processing pipeline.

Preprocessing is entirely automatic and can be performed once, before mesh animation occurs per frame. It can be conducted entirely by novice users interactively (the entire preprocessing phase takes about one minute total).

At runtime, KinÊtre computes deformations of the sparsely sampled graph per frame based on real-time human skeleton data captured with the Kinect. We use the publicly available skeletal tracker in the Kinect SDK (a variant of [30]), which gives a prediction of 20 joint positions on the human body. Deformations of the graph are then propagated to the higher resolution mesh. More specifically, the steps are as follows:

1. A "user in the loop" *rigging* step that renders both the input mesh and the tracked human skeleton to the user, and allows the user to pose herself such that her body intersects with the mesh. This metaphorically *attaches* the joints of the tracked skeleton to the surface of the mesh. In practice, the attachment is specified as a set of point constraints on the deformation graph.

2. When the user moves, we solve for the optimal transformations on the deformation graph that is simultaneously smooth, feature-preserving, and satisfies the user's motion constraints. This is formulated as a nonlinear energy minimization (extending [32]), which we solve for each frame in real-time.

3. The final step takes the transformations computed on the sparse deformation graph and applies them to the dense 3D mesh using an algorithm similar to linear blend skinning (LBS) [23, 26].

The following sections describe these steps in more detail.

Acquisition, Segmentation and Meshing

In the KinectFusion system [15], depth data from the camera is integrated into a regular voxel grid structure stored on the GPU. Surface data is encoded *implicitly* into voxels as signed distances, truncated to a predefined region around the surface, with new values integrated using a weighted running average [8]. The global pose of the moving depth camera is predicted using point-plane ICP [7], and drift is mitigated by aligning the current raw depth map with the accumulated model (instead of the previous raw frame).

The system produces a 3D volumetric reconstruction of the scene accumulated from the moving depth camera. To extract a specific object from this full 3D reconstruction two simple segmentation methods are provided. The first described in [15] allows the user to physically remove the desired object from the reconstructed scene. Taking the derivative of the signed distance values over time, we label regions of the voxel grid with high change. A full pass over the voxel grid extracts these labeled connected components and the largest region is chosen as the foreground object. This works well for objects that are physically small enough to be moved. For larger objects, a second method takes the current physical camera pose, raycasts the voxel grid and extracts the dominant plane using RANSAC, segmenting any resting objects (again the largest 3D connected component is assumed to be the desired object).

Next we extract a geometric isosurface from the foreground labeled volumetric dataset using a GPU-based marching cubes algorithm. For each voxel, the signed distance value at its eight corners is computed. The algorithm uses these computed signed distances as a lookup to produce the correct polygon at the specific voxel.

Now that we can acquire, segment, and generate a 3D mesh from a physical object, we next derive a new method that maps tracked user motions to mesh deformations.

Embedded Deformation

We first briefly describe the Embedded Deformation (ED) technique [32] that our system builds upon. Later sections introduce key extensions in the form of a new sampling and point constraint strategy that work on a greater variety of meshes, and a new energy function for the deformation graph. These better accommodate our scenario, which has poorly sampled geometry and a multitude of constraints from a tracked human skeleton.

Like many skinning techniques, ED assumes that shape deformations vary smoothly in space and can be expressed as linear combinations of a sparse set of affine transformations. ED models its lower-dimensional basis as a graph. The nodes of the ED graph are a sparse subset of the samples at locations $\mathbf{g}_j$ on the surface, and its edges are the k-nearest neighbors of each node in $\mathbb{R}^3$, with $k = 4$.

Each node deforms space in a region surrounding its position $\mathbf{g}_j$ with an affine transformation $A_j = (R_j, \mathbf{t}_j)$, where R is a 3×3 matrix and $\mathbf{t}$ a 3×1 vector. This affine transform maps the point $\mathbf{v}$ to:

$$A_j[\mathbf{v}] = R_j(\mathbf{v} - \mathbf{g}_j) + \mathbf{g}_j + \mathbf{t}_j \tag{1}$$

The deformation defined by multiple nodes is combined linearly as in LBS. Given a point $\mathbf{v} \in \mathbb{R}^3$, the deformation graph maps it to:

$$\phi(\mathbf{v}) = \sum_{j=0}^{k-1} w_j(\mathbf{v}) A_j[\mathbf{v}] \tag{2}$$

The blending weights $w_j(\mathbf{v})$ are nonzero only for the k nodes closest to $\mathbf{v}$, fall off linearly with distance, and sum to 1. Intuitively, if transformations defined by the nodes vary smoothly, and the nodes are of sufficient density, then the vertices should also deform smoothly.

Energy Formulation To constrain the space of transformations, Sumner et al. make a number of reasonable assumptions and formulate them as an energy minimization problem. The first term minimizes stretching by biasing the solution towards isometry:

$$E_{\text{rot}} = \sum_j ||R_j^T R_j - I||_F^2, \tag{3}$$

where I is the identity matrix and $|| \cdot ||_F$ denotes the Frobenius norm. To ensure that the deformation field is smooth, a regularization term encourages each node to map its neighbors near where they would map themselves:

$$E_{\text{reg}} = \sum_j \sum_{k \in N(j)} ||A_j[\mathbf{g}_k] - A_k[\mathbf{g}_k]||_2^2, \tag{4}$$

where $N(j)$ is the set of neighbors of j in the graph. Finally, the deformation model allows the user to add an arbitrary number of point constraints, mapping each given point $\mathbf{p}_\ell$ to a desired target $\mathbf{q}_\ell$:

$$E_{\text{con}} = \sum_\ell ||\phi(\mathbf{p}_\ell) - \mathbf{q}_\ell||_2^2 \tag{5}$$

where $\phi(\mathbf{p}_\ell)$ is the result of applying the deformation (Eq. 2) to $\mathbf{p}_\ell$. The total energy is a weighted sum of the three quadratic terms that is minimized using the Gauss-Newton algorithm.

EXTENDING EMBEDDED DEFORMATION

Although standard ED demonstrates impressive results on a variety of shapes, including incomplete polygon soups, we find in our experiments that the technique has difficulty with arbitrary, incomplete meshes that come from 3D acquisition. Dealing with such incomplete scans is key for KinÊtre.

For our scenarios, three major issues arise which require extensions to the work of Sumner et al. [32]. The first is the quality of the deformation graph: standard ED uses Poisson Disk sampling under the Euclidean distance metric to place graph vertices [32]. This sampling strategy tends to miss areas of the surface with high curvature and leads to artifacts where semantically unrelated parts are linked. Figure 3 illustrates how the Euclidean Poisson Disk sampling fails on a scan of a chair and a 3D model of a horse. The output deformation graphs of both meshes have connections occurring across separate legs. This causes undesirable effects as deformations to one leg will influence the other. We address this issue with a new sampling strategy based on a 5D orientation-aware distance metric.

The second issue is graph connectivity: the k-nearest-neighbor deformation graph is not guaranteed to be connected. Figure 4 (labeled a-c) shows one illustrative example, where one of the back legs of a horse model is completely unconnected to the rest of the deformation graph. This leads to unnatural

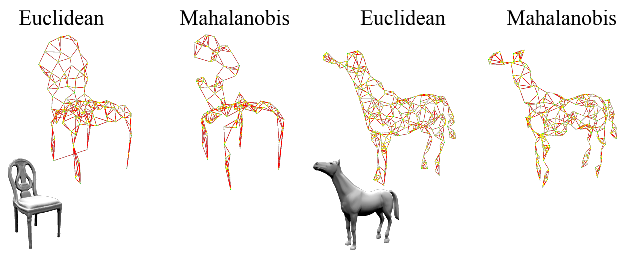

Figure 3: Orientation-aware sampling. With a Euclidean distance, samples are better distributed over the surface, but semantically separate parts tend to be connected. With a 5D Mahalanobis distance, spurious edges are reduced by placing more samples near areas of high curvature.

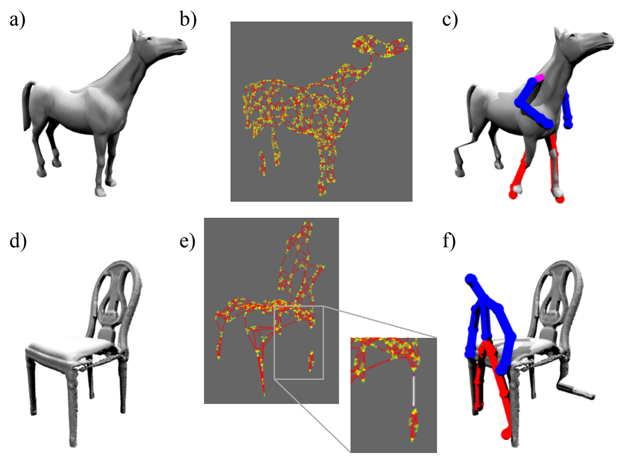

Figure 4: Mesh connectivity and regularization. Top row: a) Input mesh. A disconnected deformation graph b) leads to portions of the mesh not moving with the user and shared regions collapsing unnaturally c). Bottom row: d) Input mesh. e) Disconnected graph connected by spanning tree algorithm (inset). f) Without the rigidity regularizer, the remaining degrees of freedom allows the chair leg to rotate.

deformations in response to user motion. While not considered in [32], in practice these unconnected graphs occur frequently given input scan data or even complex models (such as the horse in Figure 4). We therefore address this explicitly in our sampling strategy.

The final issue arises due to under-constrained parts of the generated deformation graph: even when connected, the graph topology often leads to islands of under-constrained nodes. An example is shown in Figure 4 (labeled d-f), where the chair leg is connected through a single graph edge to the main body of the chair. This results in the chair leg being able to rotate unnaturally, independently of the deformations on the main body. We address this issue by applying an additional rigidity regularizer to these parts of the graph to ensure a well-posed optimization.

Orientation-Aware Sampling We ameliorate the sampling problem near areas of high curvature by adapting the strategy used by Lehtinen et al. [24] for generating Poisson Disk patterns on unstructured geometry. Given surface samples $\mathbf{p}$ and $\mathbf{q}$ with normals $\vec{n}_p$ and $\vec{n}_q$, we can define a 5D orientation-aware distance $D(\mathbf{p}, \mathbf{q})$:

$$\mathrm{dist}_q(\mathbf{p}) = ||\mathbf{p} - \mathbf{q} + 2((\mathbf{p} - \mathbf{q}) \cdot \vec{n}_q)\vec{n}_q|| \qquad (6)$$

$$D(\mathbf{p}, \mathbf{q}) = \frac{\min(\mathrm{dist}_q(\mathbf{p}), \mathrm{dist}_p(\mathbf{q}))}{\max(0, \vec{n}_p \cdot \vec{n}_q)}$$

Intuitively, the Mahalanobis distance $\mathrm{dist}_q(\mathbf{p})$ transforms the spherical isosurfaces around $\mathbf{q}$ defined by the Euclidean distance into ellipsoids with the $\vec{n}_q$ axis squashed by a factor of $1/3$. It increases the distance of points that deviate from the plane tangent to $\mathbf{q}$. The symmetric distance $D(\mathbf{p}, \mathbf{q})$ has the nice property that when it is used for Poisson Disk pattern generation, it allows differently oriented samples to be placed arbitrarily close together. We generate such a pattern as the node positions of our deformation graph, with k-nearest neighbor edges defined under the same metric. Since we typically want a fixed framerate, we choose a fixed number of samples N and define the minimum separation radius $R = \sqrt{A/N}$, where A is the total surface area. We use $N = 128$ in our implementation. Alternatively, one can also provide R defined by desired level of detail and vary the number of samples. Actual generation is performed by brute force dart throwing. Finally, the surface normal is stored as part of the graph node for later use in detecting how to attach a user's skeleton to the graph.

Figure 3 shows our Mahalanobis-based sampling strategy compared with the Euclidean-based approach of [32].

Mesh Connectivity and Regularization It is easy to see that a k-nearest neighbor graph is not necessarily connected. The lack of scan data or insufficient sampling can lead to a graph composed of several connected components. Unless there is a user constraint attached to each component, the energy is underconstrained. To illustrate, in the first row of Figure 4, the small disconnected graph component of the horse's rear leg is not influenced by the user constraints and remains stationary. While the rest of the horse moves with the user, the vertices of the leg stays in place and vertices shared by both components collapse unnaturally. We solve this problem by connecting the graph, and guarantee a well-posed optimization problem by introducing an additional regularization term.

We connect the original k-nearest neighbor graph $G = (V, E)$ by first making it undirected: if B is a neighbor of A, then ensure that A is also a neighbor of B. We then compute its connected components and form a new complete, weighted graph $G' = (V', E')$, whose nodes V' are the connected components of G, and every pair of nodes (u', v') has weight equal to the minimum distance between any two nodes $s \in u' \subset V$ and $t \in v' \subset V$. We then compute a minimum spanning tree on G', and add the (undirected) edge (s, t) to E if there is an edge between their connected components in the spanning tree.

A connected deformation graph alone is insufficient to ensure a stable deformation. In particular, the standard ED energy function contains a number of gauge freedoms due to symmetry: a subgraph without user constraints, weakly connected to the rest of the graph through a single *cut vertex* (a vertex whose removal increases the number of connected components), is free to rotate about that point. These cut vertices include the ones where we added additional edges above (see Figure 4, second row). Therefore, at these edges, we enforce a stronger regularization condition:

$$E_{\text{rig}} = \sum_{(j,k)} ||R_k^T R_j - I||_F^2 + ||\mathbf{t}_k - \mathbf{t}_j||_2^2 \qquad (7)$$

The *rigidity regularizer* says that if a piece of the mesh is only loosely connected, the bridge edge (j, k) encourages it to be coupled rigidly with the rest of the mesh. Because this is a stronger condition than the basic regularizer (12 constraints rather than 3), we use a smaller weight. The final energy is then:

$$E = w_{\text{rot}} E_{\text{rot}} + w_{\text{reg}} E_{\text{reg}} + w_{\text{con}} E_{\text{con}} + w_{\text{rig}} E_{\text{rig}} \qquad (8)$$

We use the same weights as [32] for the standard ED terms, and set $w_{\text{rig}} = w_{\text{reg}}/10$.

Constraints

The deformation model supports placement of arbitrary point constraints in space. A basic approach to attach a tracked skeleton to the surface is to simply make each joint a constraint. At attachment time, the position of each joint $\mathbf{p}_\ell$ is stored along with Euclidean distances to the k-nearest nodes in the graph. As the user moves, each joint moves to a new position $\mathbf{q}_\ell$. We update the energy such that the k nearest nodes map $\mathbf{p}_\ell \to \mathbf{q}_\ell$.

In our experiments, we found that simply attaching joints as point constraints is inadequate for two reasons. First, even with multiple depth cues, the user is not very good at judging whether she is intersecting accurately with the mesh and often places her limbs outside the surface. Secondly, using only two samples on longer limbs such as the femur is insufficient for the deformation graph to faithfully reproduce more complex motions. Therefore, we elected to place additional constraints on bones.

We model each bone of a tracked skeleton as a cylinder with predefined radius (currently 4cm). We place 16 regularly spaced samples along the length and circumference of each bone and also calculate their outward facing normal. We allow a bone sample to be attached to a graph node if it is within 10cm, and the normals are less than 90 degrees apart. Using normals makes our system robust to small misalignments between the user and the surface. For example, if the user is animating another scanned human, her arms may not be perfectly aligned with those of the scan. The surface normal test lets the front of her arm attach to the front of the arm in the scan, and likewise for the back. In practice, this works well with one-sided surfaces that come from KinectFusion.

KINÊTRE IN ACTION

We implemented a prototype system called KinÊtre, which lets users acquire 3D scans and rapidly animate them using their body. Our system also permits existing higher-quality 3D models to be imported and again, animated using human motion. The animations occur in real-time as demonstrated in the accompanying video. We stress that none of the meshes are rigged before being "possessed" within KinÊtre.

Non-Human Animation Figures 1 and 5 illustrates KinÊtre operating on a variety of non-humanoid shapes. In Figure 1, a chair is scanned by the user, automatically rigged and animated when he places his two feet and spine to approximately correspond with the back of the chair. Figure 1 shows how the mesh deformations correspond to the motions of the body. Notice how the deformations propagate across the mesh, so that even when only the back legs of the chair are rigged the front legs move in concert.

Figure 5 (bottom right) shows another example, animating a scan of a step ladder. Our system also works with more detailed closed 3D models, such as the bookcase, which once attached, can start walking, bending or even kicking based on the motion of the user (see Figure 5).

We believe that these types of scenarios where users can animate non-humanoid everyday objects will capture people's imagination, particularly young children. With KinÊtre, children can tell stories where they possess different non-human characters, and record them as home CG movies.

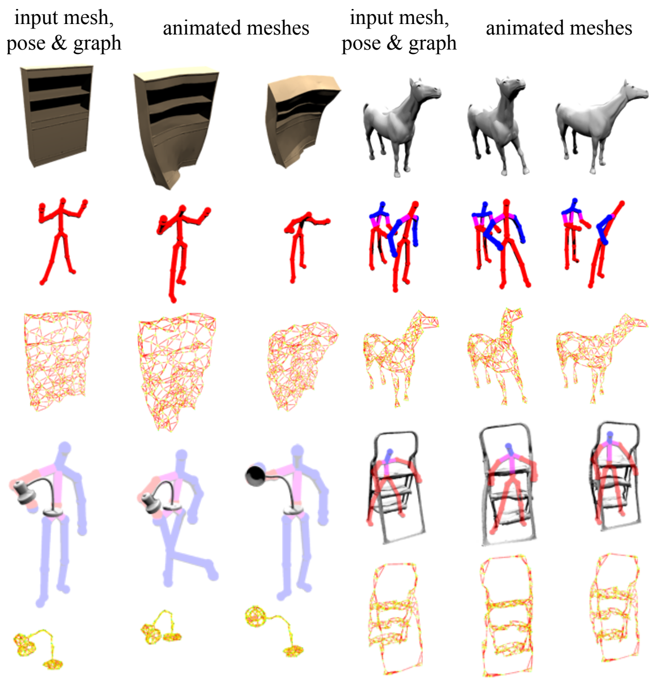

Figure 5: Gallery of non-humanoid animations. Clockwise from top left: making a modeled bookcase walk, two users controlling a horse, a jumping stepladder, and using only the hip and right arm to reenact Pixar's Luxo Jr.

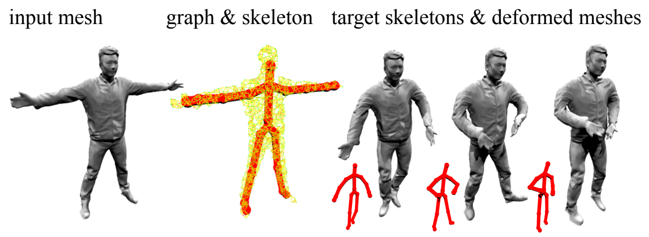

Figure 6: Animating a human scan for realistic avatars. The user can directly embed his skeleton inside the scanned mesh (left and center). Tracked skeletal motions produce realistic mesh deformations (right).

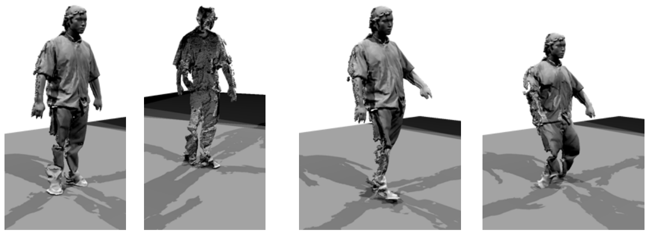
a) partial scan of human b) robust interactive deformations

Figure 7: A partial scan of a human with the back of the mesh completely missing (a). User embeds herself and can still animate the mesh in realistic ways (b).

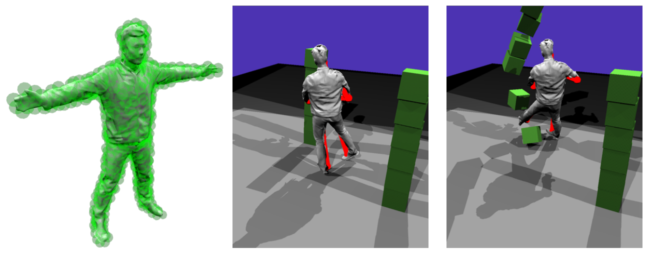
a) sphere proxies b) interactive physical simulation

Figure 8: A user mesh approximated as a series of physics-enabled kinematic spheres (a). This representation allows open ended interactions between the mesh and a virtual physics enabled scene (b).

Possessing and Animating Human Avatars Another key application area for KinÊtre is in creating more realistic avatars for gaming and teleconferencing scenarios. For example, an entire 360° reconstruction of the human body can be acquired, segmented from the ground plane and meshed using KinectFusion in less than a minute. Figure 6 shows an initially static scan of the user's body. Once acquired, he can animate his avatar, and our results show that our method supports fairly large mesh deformations beyond the original pose. *Possessing* one another's avatars can also become a playful activity: KinÊtre treats scans of humans like any other mesh and any user can control the avatar.

For teleconferencing, studies have shown the added benefits of avatar realism for collaboration [11]. KinÊtre is obviously agnostic to whether meshes are textured, and the reconstruction system could be used to create RGB textured meshes for added realism. For gaming, users could control their own avatar for a more immersive experience. Simple 3D modeling techniques such as those demonstrated in [12] could be incorporated if users want to add fun effects or personalize their characters.

Partial Scans One key feature of KinÊtre is to support animation of partial scan data. In Figures 1 and 5 both the chair and step ladder are not closed meshes with incomplete geometry. Figure 7 illustrates KinÊtre's robustness to partial data. The entire back of the mesh is clearly missing and the surface has a number of holes on the lower body. Unlike other animation systems such as [4], we demonstrate how our method can still produce a compelling deformation result even with missing data. These types of partial scans failed with our initial implementation based purely on Embedded Deformation [32]. Our extensions are therefore critical in ensuring that these extreme types of partially scanned meshes (which will be common case in everyday use) can be brought to life.

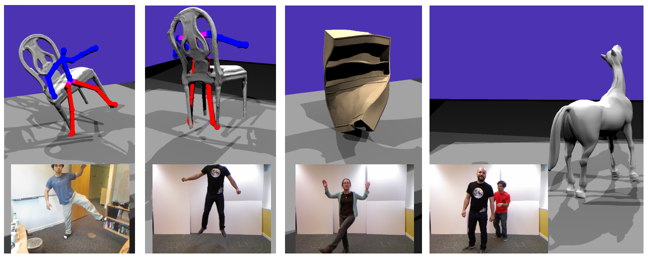
Figure 9: Users interacting with KinÊtre during informal qualitative sessions. From left: users bind to different parts of a chair to make it walk and jump. A user animates a bookcase. Two users coordinate to control the motion of a virtual horse.

Other Animation Scenarios We demonstrate some more sophisticated usage scenarios. In Figure 5, two users are simultaneously controlling both pairs of legs of the horse, as well as its head and back. This requires only simple additions to the existing pipeline to accept data from a second skeleton; our deformation method is agnostic to this change. We believe these types of multi-user scenarios will be particularly compelling in gaming or storytelling scenarios.

Depending on the scenario, it does not always make semantic sense to use all the joints of the user's body. We assign names to overlapping subsets of joints such as "left arm", "upper body", etc, which the user can toggle using the speech interface. For example, this allows the user to animate a lamp using only his hip and arm to reenact Pixar's Luxo Jr. (see Figure 5).

Physics-based Interactions Finally, once meshes are possessed by the user, they can easily be incorporated into physics-enabled interactions. For example, Figure 8 and the accompanying video show how one person can possess a friend's avatar and rampage through a physics-enabled world, knocking over towers of boxes, or how a possessed chair or stepladder can be used to play dodgeball. While these examples are simple, the combination of capturing the physical world in the digital domain, bringing it to life, and interacting with other virtual objects in physically plausible and implausible ways makes for playful new gaming scenarios.

These scenarios can be simply realized in a physics simulation just by approximating the mesh as a series of kinematic sphere proxies. We perform sphere packing as follows: repeatedly pick a random vertex (until all vertices are processed) on the mesh and place a sphere centered around it, pruning all neighboring vertices inside the sphere. The sphere radius is set to four times the mesh's mean edge length. This step is done alongside the other steps in the preprocessing phase of the pipeline. For each frame, after deformations are computed on the graph, we move the kinematic spheres accordingly. Using sphere proxies enables efficient modeling of collisions with other dynamic objects in the physical simulation, as shown in Figure 8.

Performance

We evaluated the performance of KinÊtre on a workstation with a six-core Intel Xeon W3690 processor running at 3.46 GHz with 6 GB of RAM and an NVIDIA GeForce GTX 580 GPU. With this setup, the "preprocessing" phase (scanning, segmentation, and sampling) takes ˜1 minute for a human-sized object. The sampling part runs in ˜10 seconds even on multi-million vertex meshes.

Given that the nearest neighbors in the deformation graph and their weights do not change, they can be precomputed and reused during GPU-based skinning. By keeping the deformation graph small, GPU skinning enables our system to scale to highly detailed geometry. Each time the graph transformations A_j change, we upload them to the GPU and all skinning is performed in a vertex shader.

The Gauss-Newton optimization is the performance bottleneck of our system. Our unoptimized CPU implementation, which performs an explicit sparse-sparse matrix multiply to form the normal equations, consistently takes about 0.0254 ms per iteration per graph node with a full set of skeletal constraints, scaling linearly with graph size. Most of our meshes use a deformation graph with 128 nodes. With a budget of 33 ms per frame, we limit our solver to 10 iterations. In practice, the solver converges in 6 iterations, which permits more complex models with larger graphs (256 nodes).

EXPERIENCING KINÊTRE

KinÊtre is only a prototype and we have yet to formally evaluate it. However, we have carried out a number of informal qualitative sessions where over 50 users of diverse background, age and gender have played with the system. While in no way a user study, we share some notable observations, which will hopefully inform our future designs of the system.

A series of non-human scans and 3D models were controlled by users, and animated in convincing ways; i.e., the meshes deformed but yet maintained their overall structure, and overstretching, collapsing, twisting or other mesh artifacts were minimal. As shown in RGB images in Figure 9 and the accompanying video, interacting with KinÊtre is a very physical activity. Users were literally physically animated the majority of the time. This is of course partly a feature of the body-centric input, but also an indicator that users were engaged and immersed in the experience. Although counterintuitive, users were perhaps the most animated when they were controlling non-human objects. We can speculate that the less anthropomorphic the object, the less overloaded or prescribed the methods of control, making users explore many different types of atypical poses.

KinÊtre produced some unexpected animations when users did not attach well to the mesh initially. In our sessions, we found that providing effective visualization and cues to the user is key. In our current implementation the mesh turns partially transparent when it intersects the skeleton, the attachment points are highlighted, and shadows provide further depth cues (see accompanying video). However, attachments were still problematic for users at times. This could be a fruitful area of research to understand how to better represent the lower dimensional representation and constraints to the user, without over complicating the UI. One example, could be that different body parts are locked in turn, with the virtual camera zooming in to show each region in more detail. Another idea is to combine off center and side views of the 3D model and skeleton to the user during rigging.

The other major source of unexpected deformation behavior was mainly due to the limitations of the Kinect body tracker. This was particularly evident when users tried to get into unexpected postures, such as bending to touch the floor with their hands in order to attach to all four legs of the chair simultaneously (with both their legs and arms), lying down or jumping too high. The Kinect SDK's body tracker did not support joint orientations (although this has changed recently) meaning that turning the head or twisting the forearm are not feasible. Given that our system is designed for control of arbitrary objects, a greater range of sensed poses might be a rich area for future work. One interesting area is to try to remove the requirement for a tracked skeleton altogether: because our method works by adding point constraints to the graph, data could be derived directly from the depth map.

Because our objects come from the real world and are acquired using a depth camera, they have physical units (in meters). Again in our sessions, we found direct 1:1 manipulation can be convenient for human-sized objects. However, in many cases the user did not scale correctly to the mesh (or vice versa). Currently, we scale the mesh according to an estimate of the user's physical height (using the Kinect body tracker). However, this remains an open question that begins to head into the mesh modeling arena [12], where of course natural user interfaces (such as hand-based gestures) can also play a part alongside direct animation. These types of exaggerated gestures could be used to add more extreme deformations to the mesh (for example causing stretching or twisting [17]) by selectively reweighting certain parts of the deformation graph (by varying per-node weights).

CONCLUSIONS

We have presented a novel system for using the human body to bring scanned 3D objects to life. The goal of KinÊtre is to let inexperienced home users quickly create animations and incorporate physical objects into games and other CG applications. Our system is not intended to directly create production-quality animation, but rather bring animation to a new audience. We have demonstrated how our method supports a variety of realistic deformations of human and nonhuman meshes, including 3D scan data, using a simple interface that leverages the user's body and speech. As demonstrated by our results in this paper and the accompanying video, the entire pipeline from geometry acquisition to rigging and animation, runs at interactive rates and requires no manual preprocessing by an artist.

Beyond enabling new interactive scenarios, our system makes technical contributions in the form of a robust deformation model based on the work by [32]. Our model easily handles a wide range of incomplete scans and affords real-time performance in the presence of numerous constraints coming from multiple users, all running on consumer hardware.

We believe that scanning technologies will become more prevalent in the future, particularly given the ubiquity of depth cameras and advances in vision algorithms. This enables a new breed of CG applications where the consumer is now potentially the CG modeler. The logical next step beyond *acquisition* and integration of captured real world data into consumer CG applications is bringing these virtual representations to life. Systems such as KinÊtre, will enable users to enrich games, home movies and other CG applications, by bringing playful animations of arbitrary scans to a wider consumer audience.

Acknowledgments

We thank our colleagues at Microsoft Research for their insightful comments. We are grateful for Robert Wang and Ilya Baran's assistance with numerical optimization. We also thank Emily Whiting for providing the voiceover, and actors Iason Oikonomidis and Angela Yao.

REFERENCES

1. Ahmed, N. et al. Automatic generation of personalized human avatars from multi-view video. In *Proc. VRST*, ACM (2005).
2. Allen, B., B. Curless, and Z. Popović. Articulated body deformation from range scan data. In *Proc. SIGGRAPH* (2002).
3. Anguelov, D. et al. Scape: shape completion and animation of people. *ACM SIGGRAPH* (2005).
4. Baran, I., and J. Popović. Automatic rigging and animation of 3d characters. *ACM SIGGRAPH* (2007).
5. Barnes, C. et al. Video puppetry. *ACM SIGGRAPH Asia* (2008).
6. Ben-Chen, M., O. Weber, and C. Gotsman. Variational harmonic maps for space deformation. In *Proc. SIGGRAPH*, ACM (2009).
7. Chen, Y., and G. Medioni. Object modelling by registration of multiple range images. *Image Vision Computing* (1992).
8. Curless, B., and M. Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH*, ACM (1996).
9. Davis, J. et al. A sketching interface for articulated figure animation. In *Proc. Symposium on Computer Animation*, ACM SIGGRAPH/Eurographics (2003).
10. Dontcheva, M., G. Yngve, and Z. Popović. Layered acting for character animation. In *Proc. SIGGRAPH*, ACM (2003).
11. Garau, M. et al. The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment. In *Proc. CHI*, ACM (2003).
12. Igarashi, T., S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. SIGGRAPH*, ACM (1999).
13. Igarashi, T., T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM SIGGRAPH* (2005).
14. Igarashi, T., T. Moscovich, and J. F. Hughes. Spatial keyframing for performance-driven animation. In *Proc. Symposium on Computer Animation*, ACM SIGGRAPH/Eurographics (2005).
15. Izadi, S. et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. UIST*, ACM (2011).
16. Jacobson, A. et al. Bounded biharmonic weights for real-time deformation. In *Proc. SIGGRAPH*, ACM (2011).
17. Jacobson, A., and O. Sorkine. Stretchable and twistable bones for skeletal shape deformation. In *Proc. SIGGRAPH Asia*, ACM (2011).
18. Johnson, M. P. et al. Sympathetic interfaces. In *Proc. CHI*, ACM (1999).
19. Joshi, P. et al. Harmonic coordinates for character articulation. In *Proc. SIGGRAPH*, ACM (2007).
20. Ju, T., S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *Proc. SIGGRAPH*, ACM (2005).
21. Kipp, M., and Q. Nguyen. Multitouch puppetry. In *Proc. ITS*, ACM (2010).
22. Knep, B. et al. Dinosaur input device. In *Proc. CHI*, ACM (1995).
23. Komatsu, K. Human skin model capable of natural shape variation. *The Visual Computer 3*, 5 (1988).
24. Lehtinen, J. et al. A meshless hierarchical representation for light transport. *ACM SIGGRAPH* (2008).
25. Lipman, Y., D. Levin, and D. Cohen-Or. Green coordinates. *ACM SIGGRAPH* (2008).
26. Magnenat-Thalmann, N., R. Laperrire, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proc. Graphics Interface*, Canadian Info. Proc. Society (1988).
27. Schaefer, S., T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM SIGGRAPH* (2006).
28. Shi, X. et al. Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM SIGGRAPH* (2007).
29. Shin, H. J. et al. Computer puppetry: An importance-based approach. *ACM Trans. on Graphics 20*, 2 (2001).
30. Shotton, J. et al. Real-time human pose recognition in parts from single depth images. In *Proc. CVPR*, IEEE (2011).
31. Sturman, D. J. Computer puppetry. *IEEE Computer Graphics & Applications 18*, 1 (1998).
32. Sumner, R. W., J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. *ACM SIGGRAPH* (2007).
33. Thorne, M., D. Burke, and M. van de Panne. Motion doodles: an interface for sketching character motion. *ACM SIGGRAPH* (Aug. 2004).
34. Vlasic, D. et al. Articulated mesh animation from multi-view silhouettes. *ACM SIGGRAPH* (2008).
35. Yamane, K., Y. Ariki, and J. Hodgins. Animating non-humanoid characters with human motion data. In *Proc. SCA*, ACM SIGGRAPH/Eurographics (2010).