Priority-Based Level of Detail Approach for Animation Interpolation of Articulated Objects

Antoine Abi Chakra
Xin Li

Computer Science Department
DigiPen Institute of Technology
Redmond, Washington, U.S.A.

KEY WORDS
Animation, Distance-Based, Level of Details (LOD), Articulated Character, Real-Time Simulation, Video Games

ABSTRACT
This paper presents an LOD technique applicable to articulated characters in real-time graphical simulation or video game software. This new method does not eliminate the polygon counts of the graphical model due to the change of distance to the viewer. Rather, the rendering expense is cut by the reduction of the animation interpolations between key frames based on a priority number that is calculated and assigned by taking several factors into account, including rotation, translation, scaling, number of children and depth in the hierarchical structure. An analysis to those sub-components of the hierarchical animation is conducted and an equation is established to qualify the contribution to visual effects of such animation sequence. The benchmark tests demonstrate that, under certain conditions, our approach can significantly reduce the number of interpolated with no noticeable visual artifacts.

1. INTRODUCTION
Since the introduction of 3D graphics, special visual effects have flooded the video game and graphical application. The number and types of such visual effects that can be employed in any particular game, such as particle systems, HDR, anti-aliasing, real-time shading, physically-based lighting, natural pronormal simulation, uneven surface mapping, etc., is highly dependent on the hardware configuration of the computer, including the CPU frequency, the GPU processing power, on-board memory size and bandwidth, the programmable shader units. On the other hand, the software efficiency and optimization are also the major factors contributing to the rendering capability.

With the transition from 2D to 3D, computer hardware was (and still is) being pushed more and more to produce better picture quality. Each generation of graphic cards adds new features to its predecessors, new techniques are easier or faster to implement, and the goal remains to same: squeezing more and more power juice into 3D applications. If programmers relied only on hardware power, then we would be forced to wait for the next generation of graphic cards to get faster frame rates and better looking games. Fortunately, that is not the case since there are many different software algorithms well researched and developed to alleviate hardware processing. One of such algorithms is the so-called “Level of Detail” (LOD) approach. Unlike other optimization techniques, the LOD method deals with visible polygons. After performing all other “polygon reduction” algorithms, it takes the distance that separates a visible model from the camera into consideration before rendering it. The greater the distance is, the fewer polygons of that object (such the less details) is rendered [1], [2] and [8].

A great deal of research has been devoted to this area. In this paper, we shortly mention some popular LOD methods previously done to reduce the polygon counts. Then we present in detail a new LOD technique designed to optimize the rendering process for articulated objects with the animated motions. Unlike any traditional LOD methods, our approach does not reduce the polygon counts of the object based on the distance to the viewer. Rather, the rendering time is saved by the reduction of the animation interpolations between key frames. The method is called the priority-based LOD approach since the decision of the interpolation reduction is made based on a priority number that is calculated and assigned by taking several factors into account, such as rotation, translation, scaling factor, number of children and depth in the hierarchical structure.

After the introduction section, we briefly review the hierarchical structure that is popularly used to represent articulated objects in video game software in Section 3. The degree of contribution that different transformation parameters make to the overall shape and visual effects of such articulated model during an animation sequence is examined in Section 4, which leads to the discussion on the analytic model to qualify such contribution in Section 5. In the sequential sections, we also present a method integrating this analytic model with the LOD distance as well as some implementation considerations. The result of benchmark tests and possible future research are also discussed in the paper.

2. PREVIOUS WORK
There are three main types LOD techniques.

2.1 Discrete LOD
Discrete LOD reduces the number of polygons of 3D models in a discontinuous way. The artist is responsible of creating several models for all objects, each with a different number of polygons. At run time, these models are switched depending on the distance separating them from the camera [10]. Some techniques are developed to avoid the popup effect when switching models, including late switching, hysteresis, alpha blending, etc. [4].

2.2. Continuous LOD
Continuous LOD deals with the popping problem which occurs in the discrete LOD. Instead of having several static meshes, the method uses the original mesh only and decreases object’s polygon count gradually as it gets smaller on the screen with
simplification operators such as edge collapse and vertex-pair collapse [3], [5], [6], [7] and [9].

Figure 1: Discrete LOD models

2.3 Non geometric LOD

Other non geometric-based LOD techniques can are also employed to save the rendering time by reducing the number of effects as the objects becomes smaller on the screen, such as utilizing 2D imposters to replace the 3D objects, etc

3. MOTION LOD

3.1 Hierarchy structure

Articulated models are widely used in applications and video game and other 3D graphical applications. In the articulated modeling technique, the 3D object is represented and stored as a tree-like structure. The root of the tree can have any number of children bones, which in turn are linked with a number of children bones of their own.

One technique to represent the motion of such articulated object is to associate each bone or joint of the hierarchical structure with a unique set of three dimensional transformations referred as key frames, along with a respective time stamp for each key frame. At run time, a set of transformation data (combining scale, rotation and translation) is created by interpolating between 2 key frames at each time stamp. This transformation data is often generated (or converted) into the form of matrix and concatenated with that of its children’s along the hierarchical structure. Obviously, any change in the transformation of any children bone will be propagated to all the bones located beneath that bone in the animation tree.

Obviously, the discrete and continuous LOD methods introduced in the previous sections can be applied individually to each part of the articulated model. However, the intention of our work is to enhance the optimization of the model by targeting another “expensive point” in the process, which is the transformation interpolation. This can be done by replacing the entire object by another one containing fewer bones when it becomes smaller on the screen, thus less transformation interpolations at each frame.

4. PRIORITY-BASED LOD

As explained previously, each bone in the hierarchy computes its own transformation matrix by interpolating between key frames, including scale, rotation and translation parameters. In general, the number of interpolations depends on the number of time “ticks” between the pair of key frame animations at the run-time. This number, however, can be reduced, without any noticeable visual anomaly if handled properly, as the rendered image of the model on the screen becomes smaller.

A simple method to achieve this goal is to reduce the number of interpolations for all the bones simultaneously as a hierarchical object moves away from the camera. The drawback of such naïve approach is that, since not all the bones of the object contribute to the change of animation equally, the possibility of visual anomalies introduced by this method could forbid further reduction of interpolations at the bones that contribute less to the motion behavior.

To determine the degree of contribution to the overall shape of the articulated model during an animation, we conduct an analysis in main factors such as rotation, translation, scaling, and other “states”, including the depth in the hierarchical structure and the number of children. We first examine how each factor individually affects the animation interpolation from one key frame to the next, combine all factors with different weights to form a so-called priority equation, then utilize the equation to implement a priority-based algorithm that determines the number of the interpolation frame reduction.

4.1 Rotation

In the key frame based animation, each bone has its independent rotation parameters. Assume that a bone has a rotation (R_A) of 90 degrees in 2 frames while another bone has a rotation (R_B) of 40 degrees in the same number of frames. (See Figure 3 below.) Obviously, skipping a certain number of interpolation steps from rotation (R_A) would have less visual effect than skipping the same number of interpolation for rotation (R_B), simply because the fact that the degree of transformation of rotation R_A is much greater and therefore noticeable than the one of rotation R_B.

Thus, our animation level of detail algorithm takes the rotation degree difference between consecutive key frames into consideration when determining how much a certain bone contributes to the overall shape of the animated model.

Figure 2. Hierarchical structure of an articulated object

Figure 3: Difference between bone rotations
4.2 Translation

In an animated model, each bone has its own set of translation vectors per key frame, which is independent from other bones' translation vectors. We should check the translation effect on the overall shape of the articulated model similarly to the way we checked the effect of rotating a bone has on it. Assume that a bone is translated by a vector \( T_A = <20,0,0> \) in 2 frames while another bone is translated by a vector \( T_B = <5,0,0> \) during the same number of frames. (See Figure 4). Obviously the greater the translation vector is, the more it is visible when interpolating the key frames. Therefore, skipping a certain number of interpolation steps for bone affected by the translation vector \( T_B \) won't be as noticeable as skipping the same number of interpolated frames for both scaling transformations from the bone affected by the translation vector \( T_A \). Consequently, the translation transformation difference between 2 consecutive key frames is taken into consideration when determining the degree of contribution of each bone to the shape of the animated model.

![Figure 4: Difference between bone translations](image)

4.3 Scaling

In the hierarchical model, each bone has a set of scaling values as a component of transformations. Since scaling a bone directly affects the offset of all the vertices associated with it relatively to the center, the difference between the scaling values of 2 consecutive key frames has a direct impact on the overall shape of the articulated model. Let’s assume that one bone is to be transformed by the scaling factor \( S_A = <2.5, 2.5, 1> \) during 2 frames while another bone is to be transformed by a vector \( S_B = <1.5, 1.5, 1> \) during the same number of frames. Obviously skipping the same number of interpolated frames for both scaling transformations \( S_A \) and \( S_B \) will have a different visual effect on the animation model.

![Figure 5: Difference between bone scaling](image)

In conclusion, the scaling factor difference between 2 key frames should also be taken into consideration when determining the contribution of each bone to the shape of the animated model.

4.4 Number of children

In the articulated model description, we mentioned that the bones have a set of children bones. Each one of these children bones will concatenate its parent bone's transformation data with its own, in order to generate its final transformation data. This final transformation data of the bone in question will be passed to its own child nodes that will do the same process, until this propagation traverses all the bones of the articulated model.

It is clear that the transformation matrix of any bone \( B_d \) (\( d \) is the depth of the bone \( B \)), built by concatenating its translation, rotation and scaling transformations, will be propagated to all the bones located under the bone \( B_d \). (See Figure 6 below.)

![Figure 6: Difference between numbers of Children bones.](image)

Because each bone's final transformation matrix needs its parent transformation matrix, then skipping a transformation interpolation for any bone would affect the final transformations of all the bones that are direct or indirect children of that bone. Thus we can conclude that skipping a transformation matrix interpolation of a bone having many direct and indirect children bones will have a greater negative visual effect on the overall shape of the articulated object.

The error term introduced by skipping interpolations in the hierarchical structure is relative to the total number of direct and indirect children bones and can be quantified as following:

\[
\delta B^\omega = \omega \times \phi_{Per \ Child}
\]

\( \omega = \) Number of Direct and Indirect Children of bone \( B \)
\( \phi_{Per \ Child} = \) Error factor per bone

\[
\delta B^\omega_{Number \ Of \ Children} = \omega \times \phi_{Per \ Child}
\]

4.4 Depth in the hierarchy

The higher level a bone is in the hierarchy, the greater its transformation matrix influences the rest of the model. As explained previously, the animation model is saved as a tree structure, where each bone computes its final transformation matrix by concatenating its own local transformation matrix with its parent space transformation matrix. This means that any change in the transformation of a bone \( B_d \) (\( d \) is the depth of the bone \( B \)) will have a direct impact on all children bones of \( B_d \). (See Figure 7 below.)

![Figure 7: Difference associated with the depth](image)
Having this in mind, it seems logical that skipping certain number of transformation interpolations of a bone located at a higher level in the animation hierarchy will distort the final shape of the animation model much more than skipping the same number of transformation interpolations of a bone located at the deeper level in the animation hierarchy. In other words, the error term introduced by skipping interpolations of any bone in the hierarchical structure is relative to the depth of the bone in the structure and can be quantified as following:

\[ B_d \]
\[ d = \text{Depth of bone} \]
\[ D = \text{Maximum depth of the animation} \]
\[ \phi_{\text{Depth}} = \text{Error factor per single depth difference} \]
\[ \delta_{\text{Depth}} = \phi_{\text{Depth}} \times \frac{D}{d} \]

### 4.5 Combining the factors

As analyzed above, computing how much a bone contributes to the overall shape of the articulated model is done by checking how much skipping some of its transformation interpolations affect the results of the animated model. The greater this visual effect is, the more the bone is considered “important”. Therefore, each bone should have certain priority based on its contribution when producing the final shape of the animation. This priority value determines how much a bone will “skip” its transformation interpolations. In other words, the higher the priority of a bone is, the more important it will contribute to the final shape of the animated object, therefore, the less frequent its transformation interpolation should be skipped.

Since each key frame of an animation of a hierarchical structure is independent, the current transformation depends solely on the transformations of its previous and next key frames. It implies that having a single priority for each bone does not seem to be enough. In other words, during an animated sequence, a bone can be transformed (i.e. change the orientation, size or location) in a great deal between certain key frames, while having a relatively minor transformation change between other key frames. Therefore, each key frame of each bone will have its own priority, whose value will alter from time to time during a simulation, depending on how much its transformation is “different” from the previous key frame’s transformation.

### 5. PRIORITY PARAMETERS

Based upon the above analysis, priority values are computed at run time to take into account the major factors that would contribute to the visual effect of the animation sequence once interpolated between pairs of key frames as listed below:

- Translation priority
- Rotation priority
- Scale priority
- Number of children priority
- Depth priority

This function is the governing equation when determining how and when the transformation interpolation should be skipped to save time once the animated object is distant to the camera. In practice, the priority value is also combined with a user-defined term to allow some flexibility and gain the control when the animated sequence is interpolated.

In this section the algorithm that implements this priority function is described in six steps as follows.

#### 5.1 Initialization

In order to assign a priority, some information on the entire animation and structure of the model is gathered during the load time. The process starts from the root of the articulated object and loop through all the bones. For each bone, it scans through all animation key frames in order to determine the “maximum transformations” values, including the following parameters between 2 consecutive frames of any bone in the structure:

- maximum translation difference,
- maximum rotation,
- maximum scale,
- maximum number of direct and indirect children, and
- maximum depth value of the articulated model.

Once computed, those five values are stored for later use.

#### 5.2 Translation

As mentioned previously, each key frame has its own translation vector. The length of this translation vector is computed in order to determine how much each bone is moving relatively to its parent bone. The greater the length of the translation vector, the more distant the bone is from parent. If the length of the translation vectors of the consecutive key frames is very different, then the bone is considered to be “agitated”.

Let \( T_n \) be the translation vector of the bone during the current key frame, and \( T_{n+1} \) be the translation vector of the next key frame. \( A_T \) is the difference between \( T_{n+1} \) and \( T_n \). Finally, \( l_T \) is the length of the vector \( A_T \), which is computed as follows:

\[
A_T = T_{n+1} - T_n
\]
\[
l_T = |A_T|
\]

After computing the translation amount of the bone from key frame \( n \) to key frame \( n+1 \), the length of that translation difference is compared to the animation’s current maximum translation difference and the result is saved in case for later use.

If \( l_T > \text{MaxTranslation} \) then \( \text{MaxTranslation} = l_T \)

#### 5.3 Rotation

Besides a translation vector, each animation key frame has rotation parameter, usually saved in the form of a quaternion. In order to compute the difference between two rotations, the dot product of the two quaternions representing the rotation transformation of the key frame \( n \) and \( n+1 \) is computed. The greater the value of this dot product is, the more the bone is rotating around its parent. Therefore the rotation priority for this particular key frame is relative to that quaternion dot product. If the dot product of the quaternions of any two consecutive key frames is high, then again the bone is considered “agitated”.

\[
\text{PAMAMETERS} = \text{Number of children priority, Scale priority, Rotation priority, Translation priority}
\]

\[
\phi_{\text{Depth}} = \text{Error factor per single depth difference}
\]

\[
\delta_{\text{Depth}} = \phi_{\text{Depth}} \times \frac{D}{d}
\]
Let $\overrightarrow{R_n}$ be the rotation quaternion of the bone during the current key frame, and $\overrightarrow{R_{n+1}}$ be the rotation quaternion of the next key frame. $\beta$ is the dot product result between $\overrightarrow{R_{n+1}}$ and $\overrightarrow{R_n}$

$$\beta = \overrightarrow{R_{n+1}} \cdot \overrightarrow{R_n}$$

After computing the dot product of 2 quaternions, we compare the result to the animation's current maximum rotation dot product and replace it if it is greater.

if ($\beta > \text{Maximum Rotation}$) then $\text{Maximum Rotation} = \beta$

5.4 Scale

The scaling parameter is usually saved as a 3D vector. The greater the scaling vector, the more the bone changes its geometry during the animation. If the magnitude of the scaling vectors of the consecutive key frames is very different, then the bone is considered to be very varying scale wise.

Let $\overrightarrow{S_n}$ be the translation vector of the bone during the current key frame, and $\overrightarrow{S_{n+1}}$ be the translation vector of the next key frame. $\overrightarrow{\Delta_S}$ is the difference between $\overrightarrow{S_{n+1}}$ and $\overrightarrow{S_n}$. Finally, $l_S$ is the length of the vector $\overrightarrow{\Delta_S}$.

$$\overrightarrow{\Delta_S} = \overrightarrow{S_{n+1}} - \overrightarrow{S_n}$$

$$l_S = |\overrightarrow{\Delta_S}|$$

After the scaling amount of the bone from key frame n to key frame n+1 is computed, the length of that scaling difference to the animation’s current maximum scaling difference is compared and saved for later analysis.

If ($l_S > \text{Maximum Scale}$) then $\text{Maximum Scale} = l_S$

5.5 Maximum number of children

Since this value is only related to the hierarchical structure of the articulated model, the traversal through all the key frames of the animation sequence is not necessary. However, note that, when calculating the maximum number of children of a bone, all its direct and indirect children must be counted. This can be easily done by recursively traversing the tree structure, counting the number of the direct children, and adding maximum number of children of all direct children to the total count.

5.6 Depth

The depth of a bone in the hierarchical structure is the depth of the sub-tree with this bone as the root. In other words, it can be easily computed by counting the maximum length of all paths from this node to its deepest leaf node.

if (Current Depth > MaxDepth) then MaxDepth = Current Depth

5.7 Combining Priorities

Now that all the required information on the articulated model and its animation is gathered, we can compute the per key frame priority of each bone. The per-key-frame priority represents how much the key frame in question contributes to the overall shape of the animated character. The more a bone is agitated during a certain key frame, the higher that its priority will be, and the less chance the transformation interpolation will be skipped during the animation. On the other hand, a relatively low priority will allow the bone to skip a greater number of transformation interpolations to save time.

The priority is computed using the same criteria that were used to collect the articulated model’s information, namely, translation, rotation, scaling, number of children, depth. A user-defined priority is also added to the final priority value. Six variables are used to represent the values as follows:

$$P_T = \text{Translation Priority}$$

$$P_R = \text{Rotation Priority}$$

$$P_S = \text{Scale Priority}$$

$$P_D = \text{Depth Priority}$$

$$P_{Num} = \text{Number of Children Priority}$$

$$P_{User} = \text{User-Set Priority}$$

$$P_{bj} = \text{Priority of key frame j of bone j}$$

In order to compute the value of each sub priority, the ratio between the key frame data and the maximum data that were collected in the previous step are calculated and saved. Bones that generate the highest criteria value will eventually end up with relatively high priorities, allowing them to minimize the number of skipped transformation interpolations.

Starting from the Root

For each children bone $b_i$

For each key frame $b_j$

$$P_{Tj} = \Delta_{Tj} / \text{MaxTranslation}$$

$$P_{Rj} = \beta_{bj} / \text{MaxRotation}$$

$$P_{Sj} = \Delta_{Sj} / \text{MaxScale}$$

$$P_{Numj} = \text{NumOfChildren} / \text{MaxNumOfChildren}$$

$$P_{Dj} = \text{Current Depth} / \text{Max Depth}$$

$$P_{bj} = (P_{Tj} + P_{Rj} + P_{Sj} + P_{Dj} + P_{Numj} + P_{User}) / 6$$

5.8 Weighted Average

After computing the value of the six sub-priorities, the final priority of each key frame is obtained by averaging those sub priorities. One approach is to treat these sub priorities equally by explicitly assigning the same weight for each one, i.e. $w = 1/6$.

$$P_{bj} = wP_{Tj} + wP_{Rj} + wP_{Sj} + wP_{Dj} + wP_{Numj} + wP_{User}$$

However, it can be also done differently when the user wants to emphasize on any particular sub priority. For example, if the key frames of the bones of the animated model are pre-known (with information provided by the artist) to be very different rotation wise, which means the difference between 2 consecutive
rotations is generally very high compared to the translation or scaling difference, it would be more logical to assign the rotation sub priority with a greater.

The weighted average of the 6 sub priorities allows users to have a finer level of flexibility (compared to the user-set sub priority factor) when determining the final sub priority of each key frame of the animated model's bones. In practice, we define the following six weight factors with different values. The sum of all the weights is equal to 1.0

\[ w_T = \text{Weight of the Translation sub-priority} \]
\[ w_R = \text{Weight of the Rotation sub-priority} \]
\[ w_S = \text{Weight of the Scale sub-priority} \]
\[ w_{Num} = \text{Weight of the number of children sub-priority} \]
\[ w_D = \text{Weight of the Depth sub-priority} \]
\[ w_{User} = \text{Weight of the User-set sub-priority} \]

It's the user's responsibility to adjust the weights of each sub priority to allow the animation level of detail algorithm to adapt to different circumstances. For example, if the animation manipulates the scale factors more frequently with greater differences than it manipulates the translation factors, then the weight associated with the scale sub priority should obviously be greater than translation's weight factor.

6. APPLYING PRIORITY

6.1 Averaging priorities

The static priority of each key frame for bones is used to manipulate the LOD. Although the entire hierarchy's information is considered to compute the static priorities, the LOD of each bone is manipulated separately. As the first step, the entire tree is traversed and the following information is gathered:

- MaxTranslation: Maximum translation during one frame
- MaxRotation: Maximum rotation during one frame
- MaxScale: Maximum scaling during one frame
- MaxNumberOfChildren: Maximum number of children bones per bone
- MaxDepth: Maximum depth of the animated model

In step 2, the algorithm steps through all key frames of each bone to compute the sub-priorities described in Section 5.7. Finally, the sub priorities are averaged in order to determine the final static priority of each key frame of each bone.

6.2 Applying with Distance

The idea of any LOD technique is to reduce the details that can no longer be perceived. This occurs when the object's final projected size of the screen is relatively small compared to its original size. In a 3D scene, this occurs when the distance separating the object and the camera increases. Since object's locations in the 3D scene vary unpredictably in video games, the final level of detail is frame-based. In other words, it is totally independent from the previous and next frame's level of detail.

In order to use the object's position when computing its current frame's level of detail, we need to specify a distance range: [MinDistance – MaxDistance]. This LOD range is relative the viewer, which is computed for each object as dynamic ratio \( \Omega \).

If the object's position is less than the lower bound of the distance range, then it is considered to be greatly visible. In this case, every bit of its detail is perceived. Therefore, any object closer to the viewer than the range's lower bound won't have its level decreased, which means all of its bones will interpolate their transformations normally as they do if there wasn't any level of detail technique applied. That is: \( \Omega = 1.0 \)

If the object's position is greater than the upper bound of the distance range, then it is considered to be barely visible, and all of its original detail is barely perceived, if even at all. Therefore, any object whose distance separating it from the viewer is greater than the upper bound of the LOD range will skip all of its transformation interpolations. That is, \( \Omega = 0.0 \)

Finally, if the distance separating the object from the viewer is within the level of detail range, then the object dynamic ratio is computed as follows:

\[ \Omega = \frac{\text{Current Distance} - \text{Minimum Distance}}{\text{Maximum Distance} - \text{Minimum Distance}} \]

It ensures that the closer an object is to the viewer, the greater its dynamic ratio will be. On the other hand, its dynamic ratio will converge to 0 as it goes away from the viewer. With the per-frame dynamic ratio of each animated object calculated, we can finally compute each bone's current level of detail by scaling its current frame's static priority based on its dynamic ratio.

For each bone: i

1. Get current key frame: j
2. Get static ratio of key frame j: \( P_i^j \)
3. Scale static priority by dynamic \( \Omega \): \( P_i^j \times \Omega \)
4. Current LOD = \( P_i^j \times \Omega \)

As the final step, each bone's current level of detail is calculated with consideration of its current key frame, its static priority and the object dynamic ratio. This allows to manipulate the number of steps this bone's transformation will be interpolated.

Remember that, in order to decrease the computation time when updating the animated model, some of its bones' transformation interpolations are to be skipped according to the bones current static priority (based on current key frame) and the object dynamic ratio (based on object's distance from the viewer). An articulated model is updated based on the time elapsed from the last time it was updated. Therefore, this level of detail algorithm uses the time factor in order to decide if the current transformation interpolation of a certain bone should be skipped.

The final priority of the bone represents the amount of time the update algorithm waits before actually skipping this bone's transformation interpolation. In other words, the current level of detail value of each bone will be used as the new time step for updating that bone's transformation matrix.

Example: Comparing 2 bones:

- Current level of detail of \( B_1 \) is 0.048
- Current level of detail of \( B_2 \) is 0.032

Notice that at run time, each bone is independent from all other bones, and it will skip its transformation interpolations using its own level of detail value. In the example above, the application
is assumed to be running at 60 frames per second, therefore the
duration of each frame is 0.016 seconds or 16 milliseconds.
Since Bone $B_1$ has a higher priority, it skips an interpolation each
48 milliseconds, comparing bone $B_2$ with a relatively lower LOD
value, which skips an interpolation each 32 milliseconds.

7 TEST AND BENCHMARKS

7.1 Performance gain
To verify the result of the animation LOD technique, the
following test is conducted with 100 animated models, each
made out of 153 bones and positioned in uniform grid. The level
of detail distance range for this test is [200 – 1700]. That is, no
level of detail will be applied to any model whose distance to the
viewer is less than the range's lower bound (200 units). On the
other hand, if that distance is greater than the range's upper
bound (1700 units), then the model will skip all its
transformation interpolations. Finally, if that distance is within
the level of detail range, then a dynamic ratio will be computed
and multiplied by each bone's current priority in order to
generate that bone's current level of detail.

Note that when no LOD is applied, the frame rate is always fixed
(At 12 frames per second in this test), because each of the 100
animated model is interpolating all its bones’ transformations.
However, when the application is set to take advantage of the
animation level of detail optimization technique, its frame rate
starts increasing when the distance separating the camera from
the animated models becomes greater than 200.

An extreme gain in performance is noticed when the distance
separating the camera from the animated objects becomes
greater than 1400~1500. The one and only reason behind this
fact is that at that distance, some of the objects are beyond the
upper bound of the level of detail range and therefore all their
transformation interpolations are skipped. The following figure
illustrates the performance gain and comparison with the animation LOD approach.

![Figure 8](image_url)

**Figure 8**: Frame rate increases when the distance separating
the objects from the camera increases

In the next test, we use 100 of same animated models and place
them at random locations in the scene. The same LOD range of
[200 – 1700] is applied and the frame rates are recorded when
the camera is zoomed in and out.

![Figure 9](image_url)

**Figure 9**: Objects are positioned as a cube

We can see that in this formation, the frame rate of the application is always above the normal frame rate where no level
detail is applied. This is due to the fact that some of the
animated models are already far from the camera. Notice that the
frame rate drops a little bit when the distance separating the
camera from the center of the group increases. It is logical since
at this point more objects are becoming visible. Animated
models skip all their transformation interpolations when they are
visible from the viewpoint. However, as the distance increases,
we can see the frame rate picking up again, until it reaches its
maximum value of 43.6 frames per second, which is identical
to the previous test, because at this distance, all the animated
models are skipping all their transformation interpolations.

7.2 Visual artifacts
Artifacts start appearing when a bone’s level of detail is really
low, which means lots of transformation interpolations will be
skipped. The problem is mostly noticeable when a bone’s
animation doesn’t get interpolated for a while, while its children
bones are. This leads to a weird animated character that is
noticeably different than what the animators had in mind.

In the next few screenshots, a debugging camera is used in order
to better analyze the artifacts. The real camera distance, which is
used by our LOD algorithm to determine the level of detail of
each bone, is printed in the top left corner of the window. LOD
is only applied to the character that is rendered on the right side

![Figure 10](image_url)

**Figure 10**: On left- lagging, on right- correct

In Figure 10, the distance separating the camera from the objects
is greater than 1550, while the LOD range is [200–1700]. It
means that most LOD values are low and they’re skipping many animation interpolations. We can see that the left leg of the character on the right is lagging (affected by our LOD algorithm), while the left leg of the character on the left shows the correct posture.

Figure 11: Both thighs’ are lagging with different error margins

In Figure 11, the distance separating the camera from the objects is still great at 1530, and the same LOD range is used: [200 – 1700]. We can see that the left legs of both characters are lagging but with different error margins.

The previous artifacts are apparent mostly when the object affected by our LOD algorithm is being rendered without taking the distance separating it from the camera into consideration, and this is basically what was done in Figure 10 and 11. A debugging camera was used in order to render the objects up close, while the real camera was only used when computing the LOD values of the characters’ bones. If the real camera is used for rendering, which is the case in any game and application, both these characters would end up occupying a very small screen surface, thus making these artifacts unnoticeable.

Figure 12: Characters rendered using the real camera

In Figure 12, the real camera was used to render both characters. Although the previous artifacts shown in Figure 10 and 11 are still there, the objects are barely visible up to the point where the artifacts are any noticeable.

8. FUTURE WORK

This level of detail approach for interpolated animations of articulated models presents a technique for reducing the amount of time needed to update animated models with minimal or no visual artifacts. The reduction decision is based on the current priority with six sub-components.

Currently these 6 sub priorities are treated equally, translating to the fact that they all contribute to the final priority value using the same weight. Alternatively, they can also be arithmetically generated by analyzing the model and collecting information on how much transformation parameters, depth and number of children bones contribute to the shape of the animated model. Base on this profiling, the level of detail algorithm can assign a different weight for each of the sub-priorities. The authors believe that it will enhance the adaptability of this LOD approach and therefore produce the better visual results.

REFERENCE


Antoine Abi Chakra is a faculty member of the CS department. He earned his B.S. degree in RTIS and M.S. degree in CS both at DigiPen, where he now teaches algorithm analysis, game implementation techniques, fundamentals of game development and portable game development. Antoine also works with an R&D team at DigiPen on various projects and is responsible for designing, implementing and delivering solutions.

Xin Li is a full professor in the CS Department at DigiPen. He has a Ph.D. degree in Computer Science with the expertise in 3D computer graphics, real-time rendering techniques, animation and modeling algorithms. Prior to his teaching position at DigiPen, Xin had six years work experience at Lockheed Martin and three years at Nintendo Software Technology, writing flight and ground vehicle simulation and video game software.