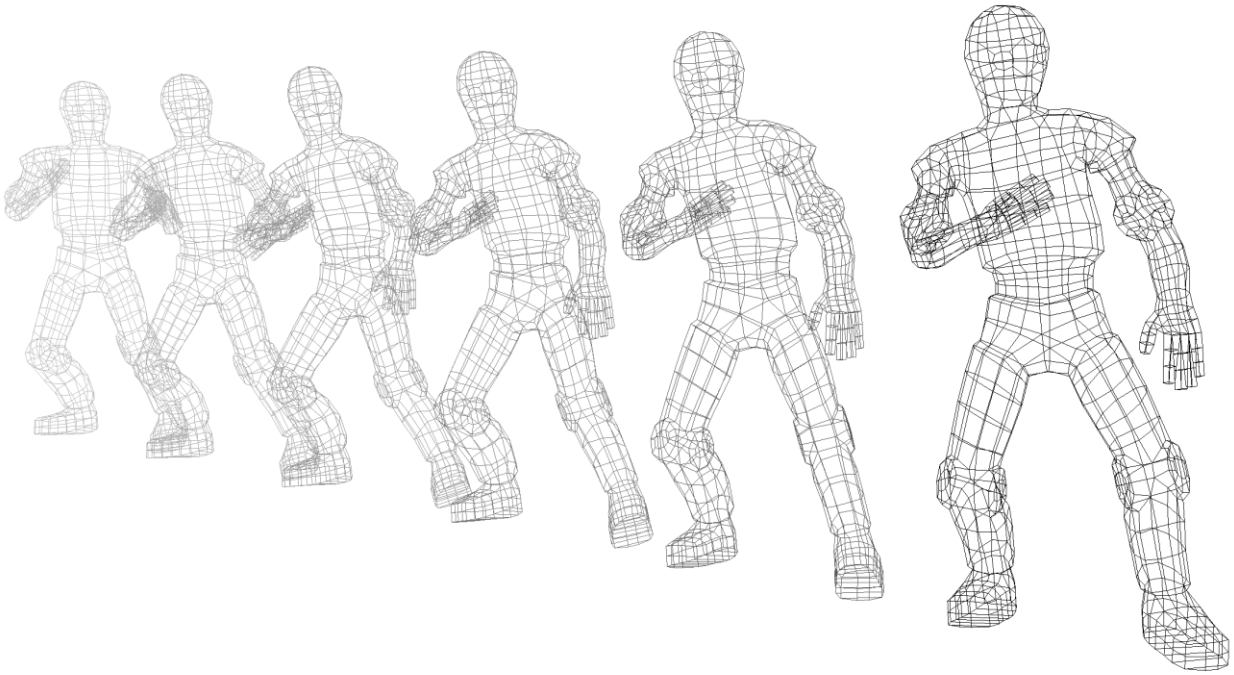
Maya 2019 - Cached Playback



Contents

Overview **3**

Migration Guide **3**

Updating to Parallel Evaluation . . . 3

1) Active Useful HUDs . . . 3

2) Disable All Performance Features . . . 4

3) Test Serial Evaluation Mode . . . 4

4) Test Parallel Evaluation Mode . . . 5

5) Enable GPU Deformation (optional) . . . 6

6) Test Interactive Manipulation . . . 6

Updating to Cached Playback . . . 6

1) Enable Evaluation Cache . . . 6

2) Enable Viewport Software Cache . . . 7

3) Enable Viewport Hardware Cache . . . 7

Key Concepts **8**

Evaluation Contexts . . . 8

Caching Points . . . 9

Caching Rules . . . 9

Custom Caching Modes . . . 10

Background Filling / Playback . . . 11

Cache Invalidation . . . 11

Evaluation Interruption . . . 12

Caching Modes **13**

Evaluation Cache . . . 14

Viewport Software Cache . . . 14

Viewport Hardware Cache . . . 15

Preferences and UI **17**

optionVars 18

General Caching Options 18

Options for the Cache Display in the Time Slider 19

Color Options for the Cache Display in the Time Slider 19

UI elements 20

Memory Limits 21

Warnings 21

Scripting 22

Plug-in Authoring **24**

Context-Specific Data 24

Testing Your Plug-ins 25

Enable Caching on Your Plug-in Node Types 25

Run Caching Validation Tests 26

Test Different Caching Parameters 27

Revisions **27**

2019 27

Overview

This guide describes Cached Playback, which was introduced in Maya 2019 as a way to boost animation performance to the next level. It describes basic concepts, ways to configure the cache as well as limitations and debugging techniques.

This guide will be of interest to riggers, TDs, and plug-in authors wishing to understand the Cached Playback architecture in order to take advantage of performance enhancements in Maya.

Even if successful Parallel Evaluation is a prerequisite for Cached Playback, this guide will not go into details on how to make a scene work with Parallel Evaluation. Please refer to **Using Parallel Maya** for more details on Parallel Evaluation.

Migration Guide

From Maya 2016, which introduced Parallel Evaluation, to Maya 2019, which introduced Cached Playback, a lot of changes were made to the core evaluation engine. As a result, some rigs and plug-ins needed to be updated in order to adapt to the new paradigm to take advantage of the new features and performance benefits.

Moving directly from a pre-Parallel Evaluation version of Maya to the latest might make it difficult to understand what needs to be updated in a given rig or set of plug-ins. This section describes a series of steps to incrementally move from the old DG-based evaluation model to the latest parallel and cache-enabled model.

Updating to Parallel Evaluation

Follow these steps when upgrading a scene from DG-based evaluation to Parallel Evaluation. These steps are not specific to Cached Playback and may have already been done if your scene runs efficiently in recent Maya versions that use Parallel Evaluation. Still, since Parallel Evaluation is a prerequisite for Cached Playback, this section is a brief reminder.

1) Active Useful HUDs

The “Evaluation” Heads Up Display (HUD) provides information about the state of the Evaluation Manager and is a useful source of feedback as you debug a rig for Parallel Evaluation.

Enable the HUD by checking the “Evaluation” checkbox in the *Display → Heads Up Display* menu.

The “Cache” HUD is also useful when working with Cached Playback. Finally, The “Frame Rate” HUD helps comparing performance between each step.

2) Disable All Performance Features

To progress incrementally through the steps, your initial state should have all performance features turned off:

- Disable Cached Playback by clicking the Cached Playback Toggle icon in the Playback Controls at the bottom right of the Time Slider.
- Disable GPU deformation by unchecking the *GPU Override* checkbox in the Animation preferences.
- Switch to *DG* evaluation mode in the Animation preferences.

Before continuing with the next steps, be sure that your scene plays without errors in the current configuration. The scene should have representative animation triggering relevant parts of the rig(s) to be tested.

3) Test Serial Evaluation Mode

The first step in detecting potential problems with the new evaluation model is to run a test in Serial Evaluation mode. This mode uses the new architecture, that is, forward evaluation, no dirty propagation at each frame, and so on, but it performs evaluation serially, in a single thread.

To enable Serial Evaluation mode, go to *Windows → Settings/Preferences → Preferences* to open the Preferences window. Then choose the *Settings → Animation* category and change the *Evaluation mode* to **Serial**.

This test detects problems caused by switching to the Parallel Evaluation model without involving concurrency issues such as race conditions.

When you run this test, that is, playback the scene in this mode, look out for:

- **Safe mode messages**: is the rig using unsupported setups? Some features, such as Motion blur, prevent access to Parallel Evaluation. Use the following command to print the reason(s) why the Evaluation Manager is disabled: `print cmds.evaluationManager(query=True, disableInfo=True)`. If anything disables the Evaluation Manager, Maya reverts to the old, DG-based, evaluation model.
- **Correctness errors**: are there any visual differences in the scene between playback in DG and Serial Evaluation modes? A typical problem might be that pieces of the scene do not move at all. This usually happens when the scene uses plug-ins that have not been updated to the new evaluation model. A common case is when the plug-in node overrides the dirty propagation mechanism (through **MPxNode::setDependentsDirty()**) without overriding the pre-evaluation mechanism (through **MPxNode::preEvaluation()**). Such nodes might use internal data members to track the status of what needs to be evaluated. This is only an example of potential problems that occur with plug-in nodes. Refer to **Using Parallel Maya** for more details.

- **Missing dependencies**: When nodes do not play by the rules and access data through means other than input attribute connections, or when they override the dirty propagation mechanism, the Evaluation Graph might miss some dependencies. Maya provides a debugging tool, only available in the Serial Evaluation mode, for tracking if evaluation occurs without the proper dependency captured in the Evaluation graph. Refer to the *Analysis Mode* section in the **Using Parallel Maya** whitepaper.

After this step, your scene should run the same in Serial Evaluation mode as it does in DG. No significant performance gain is expected at this point, since evaluation still happens serially. In some cases, this mode may even be slower than DG, since it is primarily a debugging mode.

4) Test Parallel Evaluation Mode

Once the scene runs correctly using the new evaluation model, the next step is to use this new model to enable Parallel Evaluation.

To enable Parallel Evaluation, go to *Windows* → *Settings/Preferences* → *Preferences* to open the Preferences window. Then, choose the *Settings* → *Animation* category and change the *Evaluation mode* to **Parallel**.

When running this test, that is, playing the scene back using this mode, look for:

- **Stability problems**: multi-threaded evaluation can reveal concurrency problems, such as unprotected shared data, race conditions, and so on. These problems usually manifest as crashes, but they also generate correctness errors, flickering, and so on. A common characteristic of these problems is that they show up in a slightly different way each time: the crash does not always happen on the same frame, flickering is a bit random, and so on. Multithreaded programming is a broad topic outside of the scope of this guide, but if this type of error happens in your scene, you may need to update your plug-ins so they support concurrent evaluation. You can use temporary scheduling constraints to debug and/or work around issues (see *Scheduling Type* in the **Using Parallel Maya** whitepaper).
- **Concurrency level**: once the scene runs properly, open the Profiler to confirm that the scene parallelizes well across the multiple available cores. Optimizing rigs for Parallel Evaluation is a whole topic on its own, but it is still worth a look at where the scene stands in terms of parallelism. If the scene does not parallelize well, and the performance gain is not that significant, add multiple characters to the scene and see if a higher level of parallelism is achieved.

At this point, the scene should playback well in Parallel using multiple cores, although the actual concurrency level, and expected performance gains, are highly dependent on scene content.

5) Enable GPU Deformation (optional)

Depending on the structure of deformer stacks being used, GPU deformation can provide tremendous performance gains. While GPU deformation is not a requirement for Cached Playback, it can still provide a valuable performance boost during interactive manipulation, as well as when playing back without caching.

To enable GPU deformation, go to *Windows* → *Settings/Preferences* → *Preferences* to open the Preferences window. Then, choose the *Settings* → *Animation* category and check the *GPU Override* checkbox.

The **Evaluation Toolkit** *GPU Override* section provides tools to inspect which parts of the scene are handled by GPU deformation. You can also query why some meshes were not grabbed by the GPU deformer.

Note: It may require some rig re-engineering to use GPU deformation (see *GPU Override* in the **Using Parallel Maya** whitepaper).

6) Test Interactive Manipulation

A final test is to make sure that the scene still evaluates as expected during interactive manipulation, either through Viewport manipulators, the Graph Editor, the Attribute Editor or the Channel Box.

While this test is listed as a last step, if you find problems, it might be useful to debug them by going back through the previous steps and identifying which one introduces the manipulation problems.

Updating to Cached Playback

One the rig properly works in Parallel Evaluation, you can experiment with Cached Playback.

1) Enable Evaluation Cache

Evaluation Cache is the default Cached Playback mode. Activate it by clicking the Cached Playback Toggle icon in the Playback Controls at the bottom right of the Time Slider. Right-click the icon for more options, such as setting the Caching Modes. Ensure it is set to Evaluation Cache.

If everything works properly, the cache starts filling in the background and a blue bar fills the Time Slider. The scene then plays back faster from the cache.

When you run this test, that is, play back the scene in this mode, look for:

- **Safe mode messages**: does the scene use elements that are unsupported by Cached Playback? Any messages about this display in the Script Editor as well as the Status Bar. Use this command to print messages about why Cached Playback is disabled: `print cmds.cacheEvaluator(query=True, safeModeMessages=True)`. If anything disables Cached Playback, evaluation continues to be done using the Evaluation Manager and Parallel Evaluation, but the results are recomputed every

time. The limitations are listed in **Cached Playback limitations** and **Cached Playback unsupported nodes**.

- **Stability and correctness issues**: are there any visual differences in the scene between playback with Cached Playback off or on? Does the scene crash when the cache is filling in the background? If this occurs, debug by disabling cache filling in the background. To disable cache filling in the background, in the *Caching* section of the **Evaluation Toolkit**, change *Cache fill* to *Playback*. When in this mode, Maya does not fill the cache in the background; it only fills the cache on playback. If this solves the problem, it suggests that something in the scene is not evaluating correctly in the background context. See *Plug-in Authoring* for more details.
- **Shapes not moving**: does the scene use custom plug-ins or custom shapes? By default, Cached Playback is enabled only on shapes that ship with Maya. If the scene uses custom shapes, you must activate custom caching modes for these shapes to be cached. See the *Caching Modes* section for more details.
- **Artifacts in cached data**: If you see artifacts in your cached data, use the `Flush cache` menu item to clear them out. See the *Preferences and UI* section for details about the Cached Playback Toggle menu. If you repeatedly experience the same artifacts let us know so that we can address this issue in future versions.

2) Enable Viewport Software Cache

As described in *Caching Modes*, Viewport modes provide better playback performance.

To enable Viewport Software Cache, right-click the Cached Playback Toggle icon at the bottom right of the Time Slider and choose *Viewport Software Cache*.

The expected behavior is like Evaluation Cache, that is, the cache should start filling in the background.

When you run this test, that is, play back the scene in this mode, look for the same things as Evaluation Cache: stability and correctness issues, shapes not moving, and so on. Also, your playback performance should be better with this than with Evaluation Cache. If this is not the case, use the Profiler to understand why the performance is not as expected. The *Caching Modes* section provides more details about the differences between Caching Modes.

3) Enable Viewport Hardware Cache

Once the scene is working properly in Viewport Software Cache, you can test it in Hardware mode.

To enable Viewport Hardware Cache, right-click the Cached Playback Toggle icon in the Playback Controls at the bottom right of the Time Slider and choose *Viewport Hardware Cache*.

With this mode, performance gains are expected to be the greatest of all the built-in Caching Modes, but results vary based on hardware configuration, especially in relation to your graphics card.

Key Concepts

Cached Playback introduced a few new concepts in the underlying architecture. While animators do not usually concern themselves about these, TDs and riggers will benefit from understanding this foundational information so they can get the most from the Cached Playback technology.

Evaluation Contexts

A key feature of Cached Playback is the ability to fill the cache in the background while the animator is working in Maya. This **Background Evaluation** happens in a specific evaluation context so that it is isolated from the rest of what is going on in Maya.

Evaluation contexts represent the world in which Maya performs its evaluation. Node data is kept in **Data Blocks**; each node has one data block per evaluation context. This keeps the evaluation contexts isolated so that multiple evaluations can occur without impacting each other. Think of a context as an evaluation system parameter for where and how evaluation should happen. It holds important information such as the time at which the scene should be evaluated.

The main types of Maya evaluation contexts are:

- **Normal context**: Maya's main evaluation context. It is associated with the current time on the timeline and is the one from which the Viewport queries the data to render. Evaluation in this context can be managed by the Evaluation Manager.
- **Timed context**: This type of context allows for nodes to evaluate at a time other than the current one. It allows peeking into the future or the past and can be used for effects such as delays, velocity-based tricks, and so on. It can be created using the **MDGContext constructor** taking a time as a parameter. The Evaluation Manager does **not** manage evaluation of these contexts, so pull evaluation is used instead.
- **Background context**: This type of context was introduced to support Background Evaluation. Like Timed contexts, they are associated with a time other than the current time, but evaluation with this type of context can be managed by the Evaluation Manager.

Several nodes, both built-in and custom plug-in, were written such that they did not always respect the Evaluation Context. While this previously often went unnoticed because timed context use was limited, it is no longer the case as now Background Evaluation is omnipresent. Because of this, contexts other than the normal context are constantly used.

When nodes use data blocks to read and store data, they usually respect context implicitly, since the data they use is the data associated with the current Evaluation Context. However, it is easy to stop respecting the current Evaluation Context. For example, if you store data directly on the node as a class member. Doing this causes problems because there would be one instance of this member per node, instead of per-context. See *Plug-in Authoring* for more details.

The current Evaluation Context is passed as an implicit parameter to all evaluation functions. See **MDGContext::current()** and other methods in **MDGContext** for how to work with Evaluation Contexts.

Caching Points

The Maya Cached Playback system does not need to cache every node in the scene. In most cases, only a few nodes of interest, such as geometries and transforms, need to be cached for the scene to play back from the cache without triggering evaluation. The nodes for which the caching system stores data in the cache and from which data is restored when playing back are known as **Caching Points**.

Control over what constitutes a Caching Point is done through *Caching Rules*. Nodes have different behaviors during *Background Filling vs Playback* depending on whether they are Caching Points or not.

Caching Rules

Cache configuration, that is, control over Caching Points, is done through a set of rules. An ordered set of Caching Rules is called a **Caching Mode**.

A **Caching Rule** is an expression evaluated for each node in the graph and having an effect on the cache configuration.

The most common type of rule is filter/action rules. The rule is composed of a filter that is a test performed on a given node. If the test passes, the associated action is applied. An example filter could be “is the node of the correct type” and an example action would be “enable evaluation cache”. The built-in rules, filters, and actions are described in the **cacheEvaluator** command help.

The default built-in caching modes are described in the *Caching Modes* section, but they are simply a set of rules applied sequentially to set Caching Points.

For example, the following set of rules constitutes a mode:

```
cmds.cacheEvaluator(resetRules=True)
cmds.cacheEvaluator(
    newFilter="evaluationCacheNodes",
    newAction="enableEvaluationCache"
    )
cmds.cacheEvaluator(newRule="customEvaluators")
```

The first command simply clears the set of rules. The second command adds a rule enabling Evaluation cache on all nodes of the correct types. The third command applies a safety mechanism around nodes grabbed by custom evaluators.

Viewport caching modes are slightly more complex, but the same principle applies. They start by enabling Evaluation cache, and then enable Viewport cache on the subset of supported nodes.

The Caching Rules are evaluated at graph partitioning time. Therefore, any operation that causes Evaluation Graph invalidation, such as adding new animated nodes, triggers rules re-evaluation. For example, if a new scene is opened, the current Caching Rules set is kept, but the rules are re-evaluated. Note that

built-in Caching Modes are saved in user preferences and therefore persist between Maya sessions. However, custom Caching Modes (see *Custom Caching Modes*) rules are temporary and remain active only for the current Maya session.

Custom Caching Modes

The default built-in modes (see *Caching Modes*) are meant to be generic enough to apply to a wide variety of scenes. However, you can use specific rig knowledge for greater performance.

For example, built-in modes will cache all transforms. But, a different approach would be to cache only visible transforms, or transforms that are the output of the locomotion rig. Caching fewer transforms results in a smaller memory footprint and usually provides higher playback performance, since there is less to do when restoring data from the cache.

You can use built-in filters and actions to customize caching modes by adding new rules to an existing mode or creating a custom mode from scratch. There is also an OpenMaya API that allows for the creation of custom filters. Implementing **MPxCacheConfigRuleFilter** gives a lot of flexibility: nodes can be selected based on a naming pattern, a namespace, the presence and/or value of a dynamic attribute, and so on.

You can use custom caching modes to tailor cache configuration for specific needs. For instance, plug-in shapes that are not enabled by default will require a custom mode to activate caching on the plug-in shape. The following command adds a rule to the current mode to activate Evaluation Cache on a custom shape:

```
cmds.cacheEvaluator(
    newFilter='nodeTypes',
    newFilterParam='types=+myCustomShapeType',
    newAction='enableEvaluationCache'
    )
```

Another example is to have all low-fidelity background characters cached in Viewport Hardware Cache, so that they fit in video memory, while the foreground character being edited is cached in Evaluation Cache. Any combination is possible.

However, **custom caching modes are unsupported**. They can provide a performance boost, but you may create configurations with undefined behavior. **Use at your own risk.**

Tip. Maya provides traces that you can use as a debugging tool to see the result of of caching rules evaluation.

To activate this trace, use the `dbtrace -k cacheConfig;` MEL command.

By default, a *_Trace_CacheConfig.txt* file is created with the result of rules evaluation for each animated node in the graph. Locate this file in the filesystem with `dbtrace -k cacheConfig -q -o;`.

To disable this trace, type: `dbtrace -k cacheConfig -off;`

Background Filling / Playback

Cached Playback workflows usually involve two main phases. In the first phase, the cache is filled by a background evaluation process that pre-computes values for the entire timeline, storing them in the cache. Then, playback occurs by restoring pre-computed values from the cache instead of re-evaluating them every time.

When the cache is being filled in the background, both Caching Points and non-Caching Points are evaluated in a background context. However, Caching Points require an extra step, which is copying of the evaluated values to the cache.

When playing back from the cache, Maya tries to do the minimum amount of work. Caching Points are not re-computed: data is simply copied from the cache to the node. Non-Caching Points are "skipped", that is, nothing is done to them except to leave them in a correct state so they can be re-evaluated if needed.

Cache Invalidation

Cached Playback gives you greater playback performance by restoring pre-computed data instead of re-evaluating it every frame. However, while animators work, pre-computed data becomes no longer valid because the underlying animation curves are changed. In this situation, the cache is invalidated and data must be recomputed.

Invalidation propagation starts from the node being edited and only happens when animation data is changed. Therefore:

- Changing the value of an animated attribute **does not** invalidate the cache. The temporary value on the attribute is not permanent yet, since changing the time, or re-triggering evaluation, returns the value to what is on the animation curve, and therefore in the cache. These types of temporary changes do not cause cache invalidation.

- Setting a key after editing an animated attribute, or adding, removing or modifying keys on an animation curve **does** invalidate the cache. This is because the cached data no longer represents the animation in the scene. Invalidation is triggered on the time range affected by the edit. In most cases, the affected range extends up to two keys before and after the edited key, but optimizations take factors such as broken tangents into account to invalidate a smaller range if possible.
- Changing the value of an unanimated attribute **does** invalidate the cache, as the change is permanent. Note: if the unanimated attribute does not affect anything in the animated part of the scene, the cache is not invalidated because Maya only caches what is animated. A change to an unanimated attribute affects the value for the whole timeline, so in this case, the whole time range must be invalidated.

When cache invalidation is triggered by a node being edited, it propagates to everything downstream of that node, that is, every node that depends on the edited node. All nodes reached by this propagation are marked so that they are recomputed with the next filling of the cache.

After cache invalidation, background evaluation is triggered to re-compute invalid frames. In typical animator workflows, only a small portion of the timeline needs re-evaluation. Also, only a subset of the nodes usually need to re-compute. For instance, when you animate the pinky finger of a right hand, it is likely that the face rig does not require re-evaluation, nor do other characters in the scene.

Background evaluation considers this information to evaluate the minimum number of nodes. Nodes marked as invalid are recomputed, but to do so, the nodes they depend on also must be evaluated, up to the nearest Caching Points. If the Caching Point has not been invalidated, its data is still valid and there is no need to evaluate it or anything upstream: the data can be restored and used to recompute invalid nodes downstream.

Some edits, such as adding a first key to an attribute or adding a new node, invalidates the Evaluation Graph which in turn, invalidates the entire cache. However, this does not mean that the cached values are no longer valid. The Cached Playback system tracks the topology of everything a node depends on. Once the Evaluation Graph rebuilds after being invalidated, nodes for which dependencies did not change (for instance for a character that was not affected by the edit) can have their cached values restored without needing recomputation. So, while such edits give the impression that the whole cache needs to be refilled, this happens much faster if the edit allows for re-use of the cached data.

Evaluation Interruption

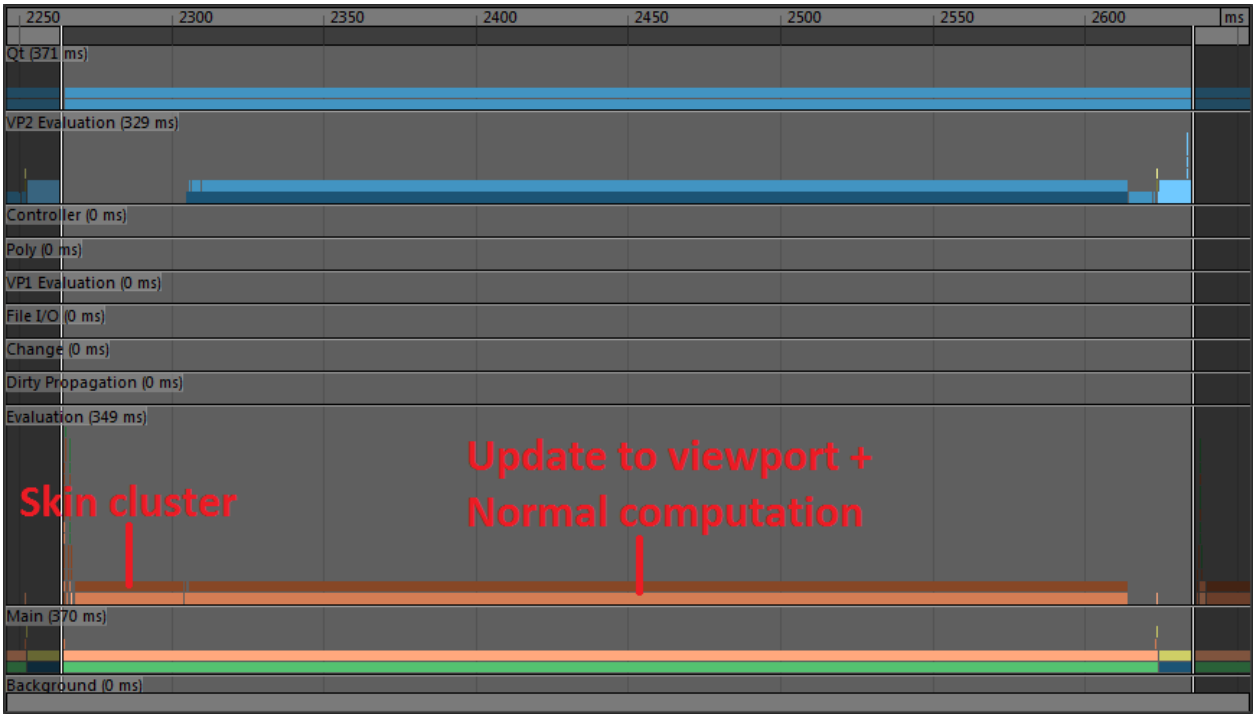
For Maya to stay responsive while the cache is filling in the background, background evaluation can be stopped in the middle of a frame. This way, all available resources (CPU, GPU, etc.) can be dedicated to foreground work. This minimizes the impact of background evaluation on animator workflows by keeping the UI responsive and interactive manipulation fast. The partially evaluated frame is discarded so when background evaluation resumes, the frame needs to be recomputed.

Tip. While the cache is filling in the background, artifacts or instabilities can be caused by interactive manipulation. You can try to work around this by disabling the background evaluation fast-interrupt feature using the `maya.cmds.backgroundEvaluationManager( interrupt=False )` command.

Caching Modes

Maya ships with three default Caching Modes. This section describes what to expect from each mode and the differences between them.

The following image is a profile of the scene used to illustrates the modes when played back using Parallel Evaluation without GPU deformation:



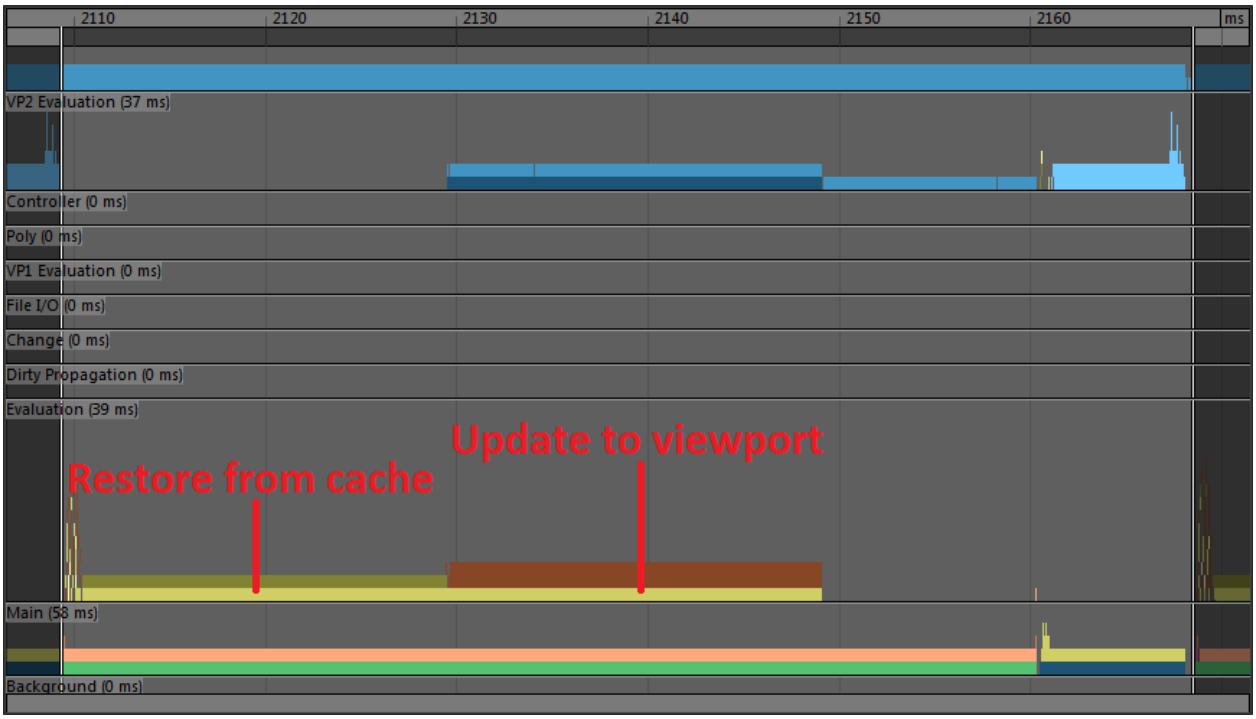
Frame time mostly consists of deformation computation and updating the Viewport representation of the geometry. Part of the translation from the data block format to the Viewport format involves recomputing normals (which are lazily computed on demand), which takes a sizeable portion of the translation time.

Let's see how Cached Playback can help performance with this scene.

Evaluation Cache

The default mode is Evaluation Cache. In this mode, computation results are cached directly in the node format, that is, the same representation as the one used in the data block. This makes this caching mode generic. It is independent of the renderer and speeds up a variety of workflows, for example, baking (since the data is already computed).

The following image is a profile of the same scene with Evaluation Cache:



The skin cluster computation is gone and replaced by a shorter section (“Restore from cache”,) where data is simply copied back from the cache instead of being fully computed.

This data still needs to be converted to the Viewport representation of the mesh, so the translation phase is still present. However, this time it is much smaller. Without caching, normals were recomputed at every frame. However, using Evaluation Cache mode, normals are only computed when filling the cache and stored in data block format. This means that while the translation will be much faster, it must still occur.

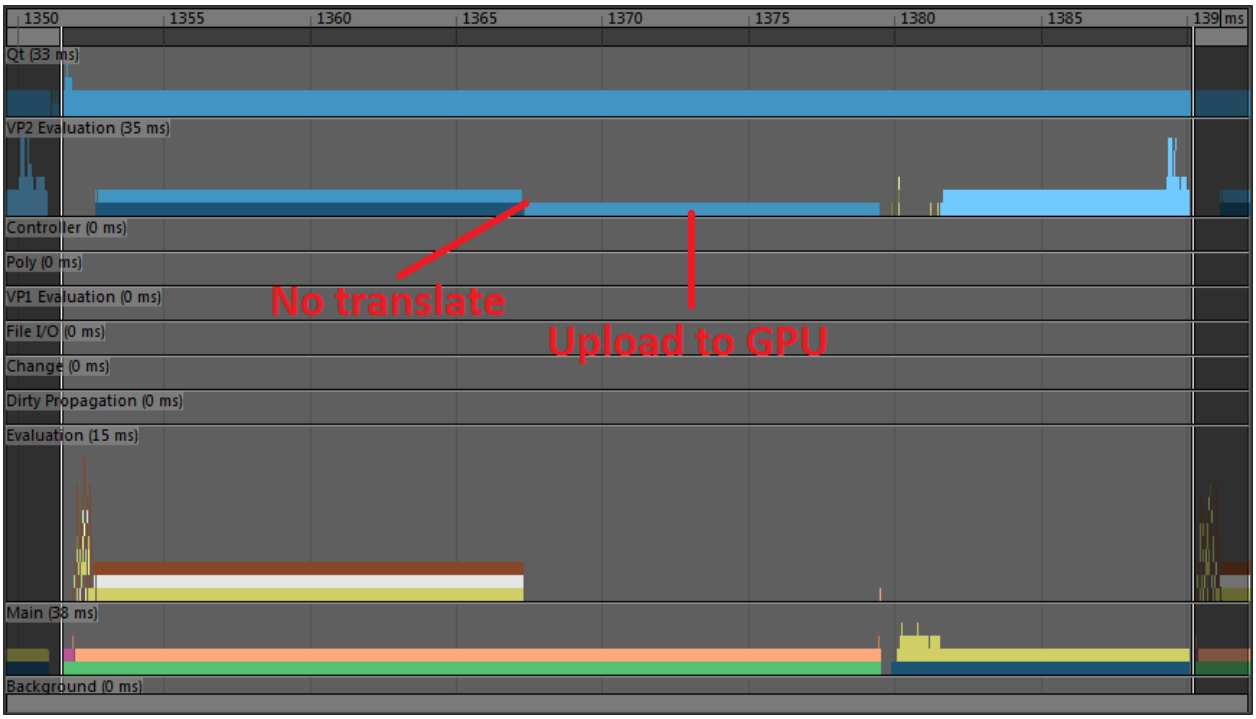
Let’s see if we can improve the situation.

Viewport Software Cache

The idea behind Viewport caches is to store cached data in a format that is ready to be consumed by the renderer, avoiding costly translation after restoring already-computed data. Instead of storing geometry

the way it is represented in the data block, it waits until after translation to the renderer buffers and stores those in RAM instead. This Caching Mode is specific to Viewport 2.0, but usually results in greater playback performance.

Here is a profile of the same scene with Viewport Software Cache:



We can now see that the “Restore from cache” section is gone, because it is no longer necessary to copy data block cached data from the cache to the node. We can directly use the ready-to-render buffers. This explains why the “translate” section of the update to viewport is also gone.

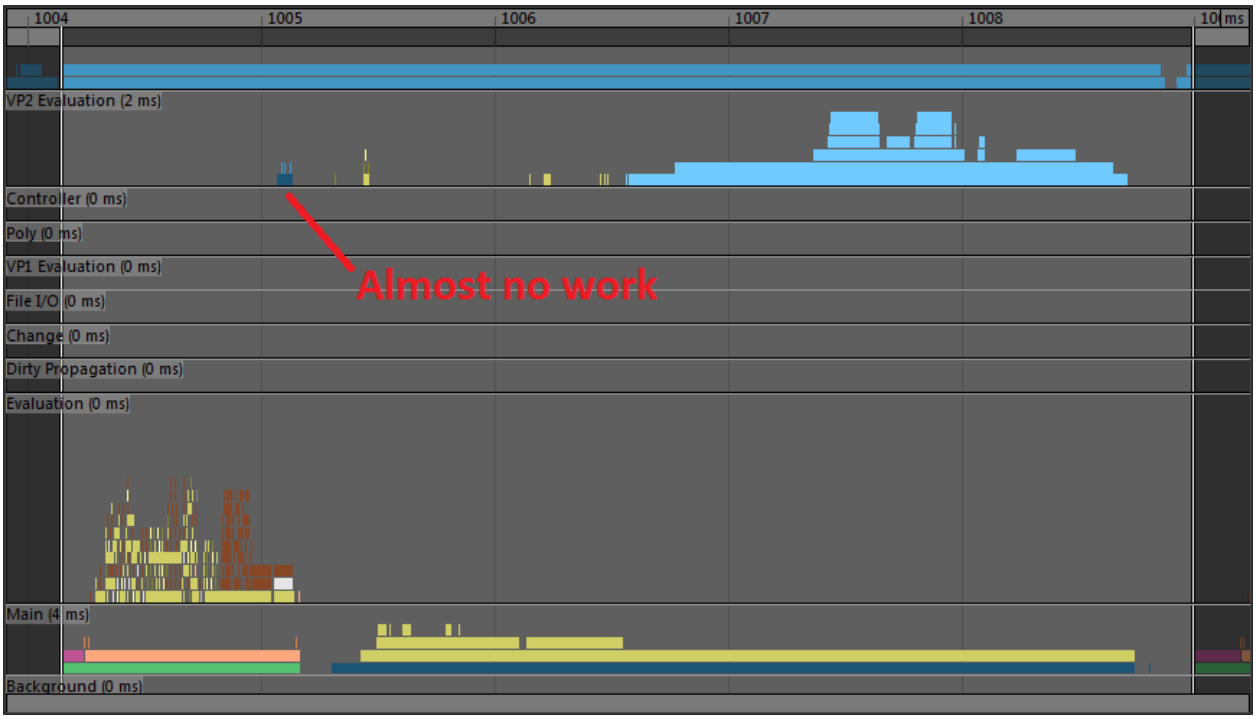
However, the restored buffers are not completely ready to be rendered. The renderer still must send those software buffers to hardware video memory.

Let’s see if we can further improve this approach.

Viewport Hardware Cache

Viewport Hardware Cache also stores buffers in the format used by the renderer, but in GPU memory instead of RAM. This is as close as we can get to having the cached data ready to be consumed by the renderer.

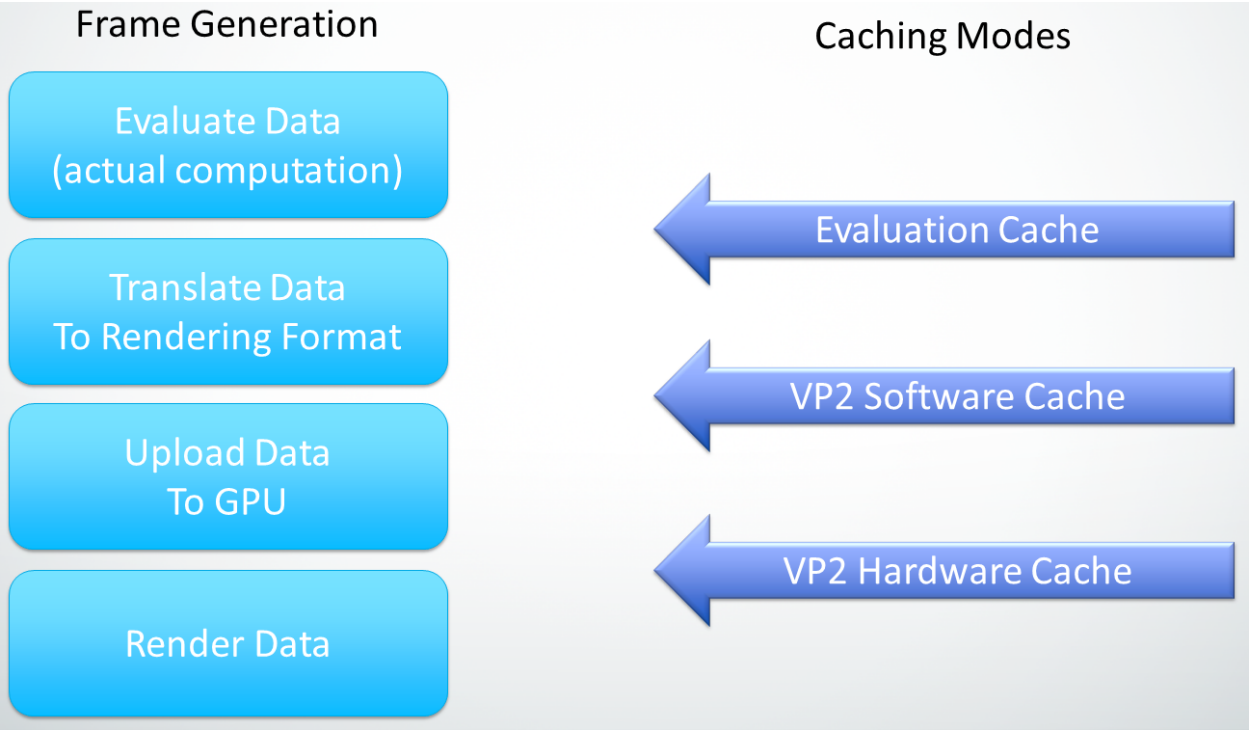
Here is a profile of the same scene with Viewport Software Cache:



In this mode, there is almost no work to be done for the Viewport. There remains a few checks to ensure the data is available, but quickly enough the renderer recognizes the data is ready to be used and renders it.

Viewport Hardware Cache provides the greatest playback speed, but it can use a lot of video memory. This is why other modes are useful: when resources are limited, compromises must be made to achieve the best possible performance with the available hardware.

The following image summarizes the steps required to generate a typical frame and where the different Caching Modes fit in the process to provide different levels of performance.



Tip. Even if the current mode is set to Viewport Software Cache or Viewport Hardware Cache, some geometries might fall back to Evaluation Cache. Viewport caches currently have limitations that force the caching strategy to resort to Evaluation Cache to avoid unsupported setups (like animated visiblity or topology), while still providing a reasonable performance gain.

You can use the `cacheConfig` trace described in the *Custom Caching Modes* section to identify nodes that switch to Evaluation Cache. You can also use the `cacheEvaluator` command to query what type of cache is enabled on any node.

You can deactivate this fallback behavior with a custom caching mode created by simply adding "fallback=0" as a *newActionParam* for the *enableVP2Cache* action in the regular Viewport Software Cache or Viewport Hardware Cache configuration rules. See the `cacheEvaluator` command help for more details.

Preferences and UI

Like most preferences, cache preferences are stored as Maya optionVars. You can identify them by name as they all have the prefix `cache`. They can be modified in one or more places in the UI, as well as with

the `optionVar` command in MEL or Python.

optionVars

Here is a complete list of all available optionVars that control caching preferences.

General Caching Options

optionVar Name	Type	Description
cacheVisibility	bool	When **true** shows the cache information HUD in the Viewport.
cachedPlaybackDiscardFramesOutOfRange	bool	When **true** proactively flushes any cached data from frames outside the current playback range
cachedPlaybackEnable	bool	When **true** use cached playback
cachedPlaybackFillDirection	string	Direction in which the cache loads in the background. Legal values: **forward** (from the current frame on), **backward** (from the current frame back), **bidirectional** (from the current frame alternating forward and backward), and **forwardFromBegin** (forward from the first frame of animation)
cachedPlaybackFillType	string	Choose when the cache is built. Legal values: **syncAsync** (during playback and in the background), **syncOnly** (only during playback), **asyncOnly** (in the background only)
cachedPlaybackHeight	integer	Defines the height of the cache status bar within the Time Slider
cachedPlaybackMemoryThreshold	float	Amount of system RAM (in percent) being allotted for Cached Playback.
cachedPlaybackMode	string	Select how Cached Playback uses system resources to increase performance. Legal values: **evaluationCache**, **viewportSoftware**, and **viewportHardware**)
cachedPlaybackPreventFrameSkipping	bool	When **true** prevents frame skipping while filling the cache in Playback mode
cachedPlaybackResourceGuard	bool	When **true** respects the memory limit and stops caching when it is reached.

optionVar Name	Type	Description
cachedPlaybackShowWarningMessages	true	When **true** displays diagnostic messages if caching is disabled

Options for the Cache Display in the Time Slider

optionVar Name	Type	Description
cachedPlaybackPosition	string	Sets the position of the cache status line relative to the timeline within the Time Slider. Legal values: **above** and **below**
cachedPlaybackShowCacheStatus	bool	When **true** displays the Cached Playback status line on the Time Slider
cachedPlaybackShowCachedSubframes	true	When **true** uses the subframe color to indicate the sections of the Time Slider with cached data at subframes
cachedPlaybackShowInvalidatedFrames	bool	When **true** uses the invalidated frame color to indicate sections of the Time Slider with invalidated cached data
cachedPlaybackShowWarningFrames	bool	When **true** uses the warning color in the cache visualization to indicate sections of the timeline where frames are not cached

Color Options for the Cache Display in the Time Slider

These are controlled with the `displayRGBColor` command. Each frame will have a horizontal bar shown in the Time Slider for various caching states at that frame. (There is only a single caching state per frame.)

Color Name	Description
cachedPlaybackCachedFrames	Color for frames with valid cached data
cachedPlaybackUncachedFrames	Color for frames without cached data
cachedPlaybackInvalidatedFrames	Color for frames with cached data that may not be valid
cachedPlaybackWarningFrames	Color for frames that were not cached due to a warning state (e.g. out of memory)
cachedPlaybackCachedSubframes	Color for frames containing valid cached data for subframes
cachedPlaybackInvalidatedSubframes	Color for frames containing cached data for subframes that may not be valid

Color Name	Description
cachedPlaybackFramesOPACITY	Opacity for colors displayed in the cache section of the Time Slider

The online help section on Cached Playback in the Time Slider gives you a detailed visual representation of where are the colors are used. Here is a quick summary:

Tip. The Time Slider's Cached Playback status line colors entire frames within the cached frame range. When playing back with fractional frames, for example stepping by 0.5 frames, the exact location of those subframes are shown as a single tick in the subframe valid or invalid color.

UI elements

The Cached Playback system uses various interface elements to control the preferences and show feedback on the current caching state.

- The **Evaluation Toolkit** contains a **Caching** section with controls for many of the preferences, as well as for doing things like flushing or invalidating the cache. There is also a **Custom Evaluators** section where you can find information about the cache, such as how many nodes are cached, on which frames, and exactly which nodes are cached.
- The **Preferences Window**, at *Windows → Settings/Preferences → Preferences*, contains controls for some of the caching preferences in the **Cached Playback** tab.
- The **Color Settings Window**, at *Windows → Settings/Preferneces → Color Settings*, contains a **Cached Playback** tab where you can set all of the Status line colors.
- The **Cached Playback Toggle** in the Playback Options provides quick access to common caching features. Click it to toggle Cached Playback, and a right-click menu provides more options. You can also access this menu from the Animation menu set in *Playback → Cached Playback*.
- The **Cached Playback HUD**, enabled with *Display → Heads Up Display* through the **Caching** checkbox. It displays the Caching Mode and the memory usage.

> **Tip.** No matter where you update your preference values; the Evaluation Toolkit, the Preferences window, the Cached Playback Toggle, or the `optionVar` command, all preferences are synchronized. That is, if the Evaluation Toolkit is open and you use the Cached Playback Toggle to enable Cached Playback, the Evaluation Toolkit state is immediately updated.

Memory Limits

The Memory Limit options let you allocate a percentage of your memory as an upper boundary for the memory the cache uses, as well as a flag to enable or disable checking that memory. Keep the memory usage of the cache within a reasonable range, such as 60-75%, to prevent cache memory from paging, to ensure optimal playback performance.

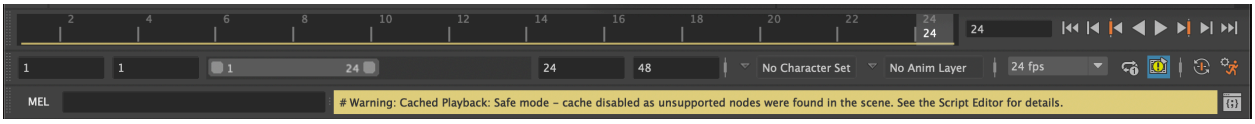
The percentage of memory use is calculated by Maya's current memory use divided by the total memory available. This memory allotment represents more than just memory used by the cache, so adjust your resource limit percentage based on the amount of memory it takes to load your scene.

When the memory resource limit is enabled and reached, caching stops and the remaining uncached frames are shown in the Time Slider in the warning color.

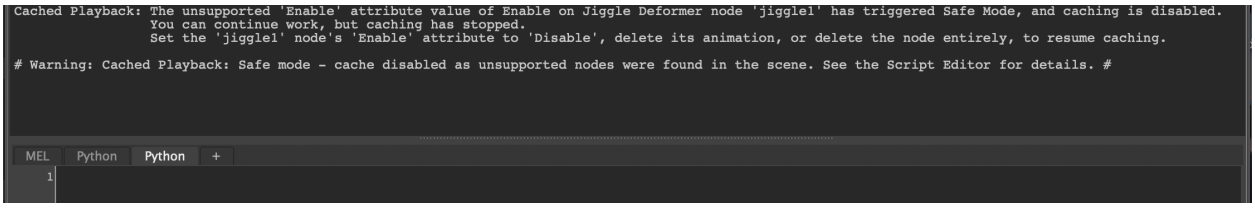
> **Tip.** Set the `cachedPlaybackDiscardFramesOutOfRange` optionVar to **true** to ensure you are not using up caching memory to store frames that are not within your current playback range.

Warnings

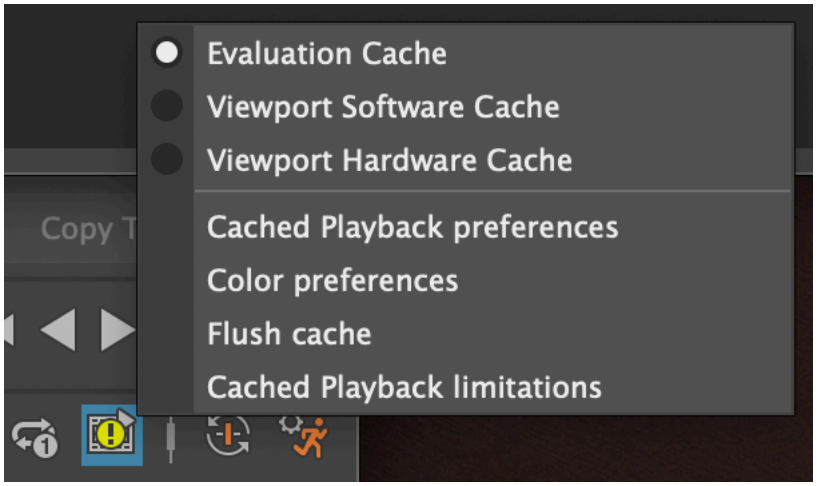
The Cached Playback system has some limitations, such as unsupported nodes. When it encounters these limitations, a yellow warning message appears in the Help line to alert you,



along with a detailed explanation in the Script Editor.



The Cached Playback Toggle also changes to the warning icon until the cause of the warning has been addressed.



Efforts have been made to prevent duplicate warning messages as they will impact performance. Warning messages may reappear whenever you make a change to the scene that could affect the cache, such as adjusting memory limits or adding new animated objects, as confirmation that the adverse effect will persist after the change.

Scripting

Cached Playback preferences can be controlled programmatically through Python.

The following example shows how to determine whether Cached Playback is enabled or not, and then enables Cached Playback.

```
from maya.plugin.evaluator.cache_preferences import CachePreferenceEnabled

# Determine whether caching is enabled or not.
if CachePreferenceEnabled().get_value():
    print "Cache is enabled"
else:
```

```
    print "Cache is disabled"

#  Enable caching
CachePreferenceEnabled().set_value( True )
```

To choose between the different built-in Caching Modes, use the CachedPreferenceMode object:

```
from maya.plugin.evaluator.cache_preferences import CachePreferenceMode

# Print the current mode.
print "Cache mode is : {}".format(CachePreferenceMode().get_value())

# To set Evaluation Cache
CachePreferenceMode().set_value( 'evaluationCache' )
# To set Viewport Software Cache
CachePreferenceMode().set_value( 'viewportSoftware' )
# To set Viewport Hardware Cache
CachePreferenceMode().set_value( 'viewportHardware' )
```

The various configuration objects are defined in the *maya.plugin.evaluator.cache_preferences* and *maya.plugin.evaluator.cache_optionvar_states* Python packages:

- *maya.plugin.evaluator.cache_preferences*
  - CachePreferenceEnabled
  - CachePreferencePreventFrameSkipping
  - CachePreferenceMode
  - CachePreferenceFillType
  - CachePreferenceFillDirection
- *maya.plugin.evaluator.cache_optionvar_states*
  - CachePreferenceHud
  - CachePreferenceResourceGuard
  - CachePreferenceMemoryThreshold
  - CachePreferenceDiscardFramesOutOfRange
  - CachePreferenceShowCacheStatus
  - CachePreferenceShowInvalidatedFrames
  - CachePreferenceShowSubframes
  - CachePreferenceShowWarningFrames
  - CachePreferenceTimesliderBarHeight
  - CachePreferenceTimesliderBarSpacing

- `CachePreferenceTimesliderBarPosition`
- `CachePreferenceShowWarningMessages`

See the *optionVars* section for a description of these parameters.

Plug-in Authoring

While plug-ins have always been required to handle evaluation in all `MDGContexts`, with Cached Playback, now it is even more important that this requirement is met. Cached Playback uses background evaluation in time-based contexts to populate the caches so any code that does not respect the current context could potentially cache incorrect data.

> **Tip.** A change was made in *Maya 2018* to introduce the concept of a current context.

You should no longer be passing around contexts or using deprecated methods that pass around contexts. If you need to reference the context explicitly, get it from `MDGContext::current()`.

Context-Specific Data

Context-specific data is defined as any information that you use in your plug-in node's `compute()` method, or other methods used to compute plug values. Normally, this takes the form of plug values within your datablock, and if possible, use this location as it will reduce the requirement for synchronization in your code.

Sometimes you may need to store data outside the datablock. For example, a complex structure that cannot be represented with attribute types. In these situations you must ensure that a uniquely identifiable copy of your data exists for each evaluation context. A simple way to implement this is to use a hash table whose key is the `MDGContext` time value, with one extra slot for the normal context, and whose values are the context-specific data in your node.

> **Tip.** When new datablocks are created for different contexts, they are initialized to the same values as the datablock for the normal context, so your context-specific data should do the same.

With caching running in background threads, it is likely that your node will be visited by different threads with different contexts. However, the evaluation engine ensures that no one node will have an evaluation request for more than one context at the same time. There should be no need for mutexes in your evaluation code, which are not recommended due to potential performance issues.

Testing Your Plug-ins

To determine if your plug-in can correctly operate in a caching environment you can use the built-in validation tools. To do this, create a representative scene that uses of your plug-in nodes, enable caching on your plug-in node types, and then run the validation tests.

Enable Caching on Your Plug-in Node Types

To enable caching for your particular node type, you must add it to the caching rules. Here is how to enable *evaluation* caching for your plug-in node `MyNodeType`.

Language	Code to enable Evaluation Cache
MEL	`cacheEvaluator -newFilter "nodeTypes" -newFilterParam "types=+MyNodeType" -newAction "enableEvaluationCache";`
Python	`maya.cmds.cacheEvaluator( newFilter='nodeTypes', newFilterParam='types=+MyNodeType', newAction='enableEvaluationCache')`

You can also use the same command with slightly different parameters to enable *viewportSoftware* caching:

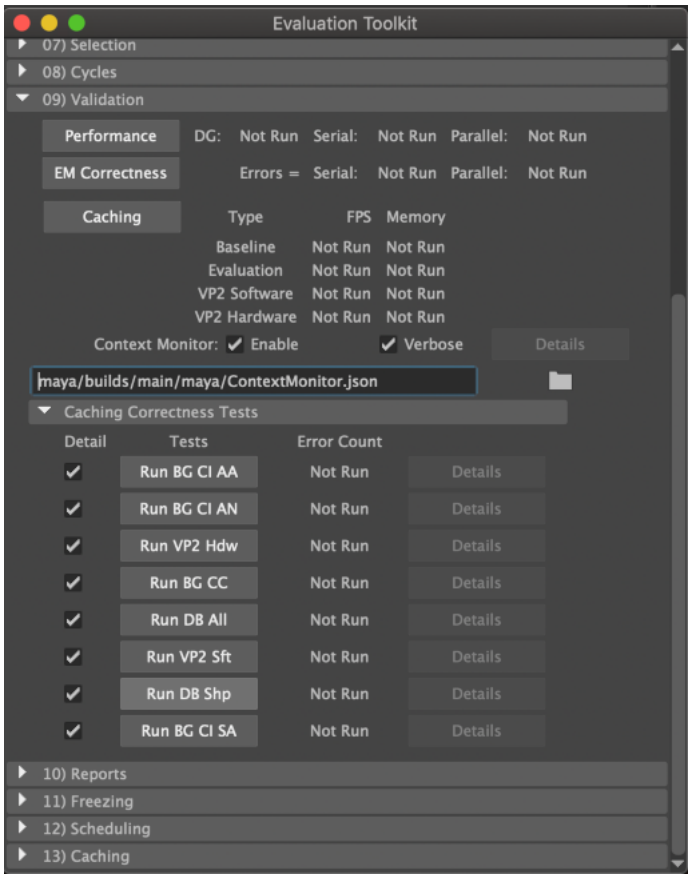
Language	Code to enable Viewport Software Cache
MEL	`cacheEvaluator -newFilter "nodeTypes" -newFilterParam "types=+MyNodeType" -newAction "enableVP2Cache" -newActionParam "useHardware=0";`
Python	`maya.cmds.cacheEvaluator( newFilter='nodeTypes', newFilterParam='types=+MyNodeType', newAction='enableVP2Cache', newActionParam='useHardware=0')`

and also *viewportHardware* caching:

Language	Code to enable Viewport Hardware Cache
MEL	`cacheEvaluator -newFilter "nodeTypes" -newFilterParam "types=+MyNodeType" -newAction "enableVP2Cache" -newActionParam "useHardware=1";`
Python	‘maya.cmds.cacheEvaluator(newFilter=‘nodeTypes’, newFilterParam=‘types=+MyNodeType’, newAction=‘enableVP2Cache’, newActionParam=‘useHardware=1’)

Run Caching Validation Tests

There are built-in tests that let you see if your nodes respect the context and will behave well in a caching environment. You can find these in the Evaluation Toolkit (*Windows → General Editors → Evaluation Toolkit*)



Create a sample scene that contains some of your plug-in nodes with animation so that you can test their operation. You can use any scene, though it is best if they resemble scenes you typically see in your environment. The only requirement is that there is animation on the plug-in nodes, as otherwise the Evaluation Manager will not cache them using the caching rules you have created.

The first set of tests measure the performance of the supported Caching Modes against an uncached baseline. To run these tests, use the **Caching** button. The frame rates and memory usage of the cache will populate the table next to the button.

Once you have your performance results, check for areas of the code where evaluation does not respect the current context. Do this by checking both *Enable* and *Verbose* options of the **Context Monitor** and rerunning the performance tests. These checks will impact performance slightly, which is why it is best to run the tests twice.

There are several different evaluation tests in the **Caching Correctness Tests** section. They ensure that the background evaluation used by caching returns the correct values. Run them in any order; they will populate the **Error Count** column when complete. Ideally run these tests with the **Detail** checkbox enabled. This way if any test returns a non-zero error count, you can use the corresponding **Details** button to show more information on the failure.

Test Different Caching Parameters

In the **13) Caching** section of the Evaluation Toolkit are several parameters that modify how caching is performed. Try changing the parameters and re-running your tests to ensure that caching behaves well in every situation. Particularly, you can lower the memory limit **% of RAM** to force the *out of memory* condition that halts caching to test playback in a mixed cached/uncached mode.

Revisions

2019

- Initial version of the document.