# Temporal Anti-Aliasing

Alexander Cech (08900070)

## 1 Overview

In this student project most of the temporal anti-aliasing (TAA) techniques described in the TAA STAR report by Yang et al. [4] were implemented. The main goal was to write a testing application that allows to experiment with different methods and parameters in varying scenes. The different methods broadly fall into two categories: Acquisition of the temporal samples, and validation and rectification of history data. The topic of temporal upsampling was also briefly experimented with, but not followed up in depth. Additionally, TAA with adaptive ray tracing, as proposed by Marrs et al. [3], was implemented.

## 2 Application Settings

As most of the command line or GUI parameters are self-explanatory or described via hints in the application itself, only the settings that require further explanation are detailed in this report. A complete summary of all parameters is given in appendices A and B.

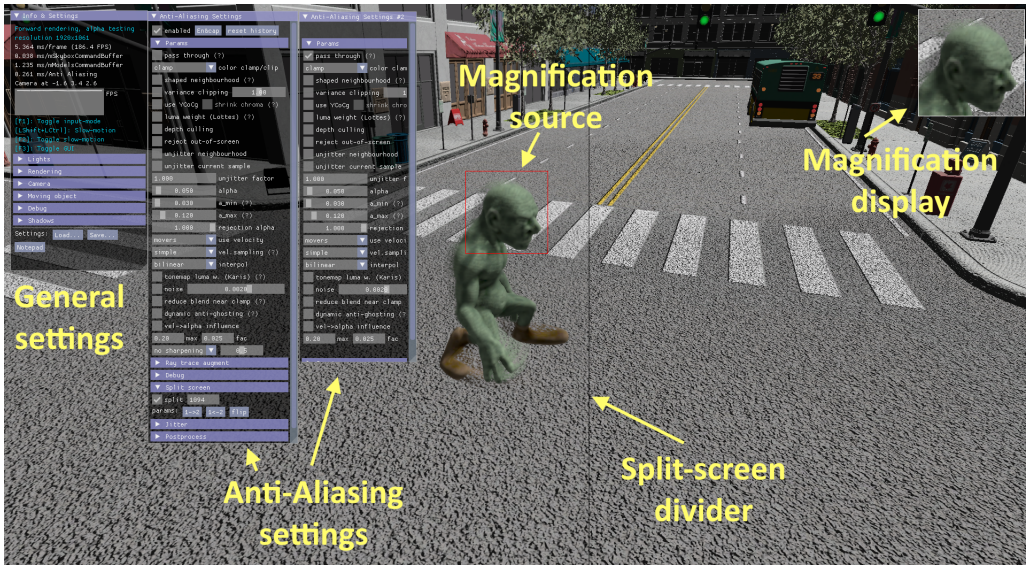


Figure 1: Screen layout

### 2.1 Command Line Parameters

The application supports a number of command line parameters (see appendix A); execute `taa -h` to get a description printed on the console. The window size or full-screen-mode can be controlled with the `-w`, `-h` and `-fullscreen` parameters. A path to the scene to be shown can be passed as final command line parameter; if omitted, the Sponza scene is used.

When experimenting with Temporal Upsampling, the parameter `-upsample <factor>` must be used, so that rendering is performed at a lower resolution than the displayed window.

### 2.2 General Settings

- In the main settings window *Info & Settings* the light sources, rendering settings (mainly for ray tracing LODs) and shadows can be parameterized in the corresponding sections.
- The section *Camera* allows to define automatic camera movement, like rotation, bobbing and strafing. Additionally a fly-through path can be edited and executed from there.

- In *Moving object* an additional (optionally animated) object that moves relative to the scene can be selected and its movement parameters defined.
- Section *Debug* holds settings for experimenting with temporally upsampling static images.
- The setting *Cap framerate* in the *Rendering* section (similar to the `-vsync` command line parameter [1]) is important for reproducibility on different computers: TAA, as implemented, is **not** frame-rate-independent[2]. A number of artifacts that are visible on lower-end machines are barely noticeable with more performant GPUs. By capping the frame rate maximum to a value both machines can handle, consistent results can be achieved.
- Automatic camera movement is quite useful to analyse motion-induced anti-aliasing artifacts and is typically used regularly when experimenting with parameters. To facilitate this, the camera *bobbing* behavior can be toggled with the keyboard key `B`.

## 2.3 Anti-Aliasing Settings

All parameters affecting the core TAA techniques described by Yang et al. [4] can be controlled in the *Anti-Aliasing Settings* window.

### Sample acquisition

- The type and sequence length of the jittering pattern, which defines the sub-pixel-shift of the rendering projection matrix, can be set in section *Jittering*.[3]
- The *alpha* parameter (in *Params*) defines the blending factor[4] between the accumulated history buffer and samples from the current frame.
- Luminance-adaptive tone mapping, as discussed in [4, Section 3.3.1], is activated with the checkbox *tonemap luma w. (Karis)*.

### History validation and rectification

- Color clamping or clipping, as well as variance clipping is available for color based rectification. Optionally these methods can be performed in the YCoCg color-space instead of RGB. Typically clamping or clipping uses the minimum and maximum color in a 3x3 pixel neighborhood. This can lead to box-like artifacts that look like low-resolution pixels when extreme local minima/maxima are present. Both *variance clipping* as well as *shaped neighborhood* can alleviate this problem. The difference is subtle: variance clipping works by using the mean and standard deviation of the neighborhood pixels to restrict the clipping box, whereas shaped neighborhood averages the extremes of a 3x3 and a 5-tap (cross-shaped) neighborhood instead. Additional details about this topic are laid out by Karis [2]. In practice both methods achieve similar results.
- *Depth culling* rejects history samples if their depth values differ too much from the expected value (which is obtained either by reprojection or velocity-vectors).
- *Reject out-of-screen* rejects samples whose history-coordinates fall outside the screen (e.g., due to lateral or rotational camera movement).
- *Use velocity* and *Vel.sampling* determine whether (and how exactly) to use screen-space velocity-vectors instead of reprojection: Using velocity can be turned off completely, activated for animated objects only, or used for all geometry. The sampling methods differ by either taking pixel-wise velocities directly, or by smoothing the vectors by examining the 3x3 neighborhood and taking either the longest or the depth-wise closest velocity vector. In practice no significant difference was observed between the sampling variants.

### Dynamic alpha

- *Luma weighting (Lottes)* automatically chooses a pixel-specific alpha value for history blending (ignoring the *alpha* setting) based on the luminance difference between current sample and history. Related settings are *a_min* and *a_max*, which define the possible range for the calculated alpha.

[1] The `-vsync` parameter lets the graphics driver fix the framerate to the monitor refresh rate, whereas the *Cap framerate* setting allows to enter an arbitrary cap, controlled by the application itself.

[2] A frame-rate-independent implementation makes little sense, since it would abandon additional quality that can be achieved by more performant machines.

[3] The additional settings in the *Jittering* section are mainly used for debugging.

[4] Actually alpha is the *inverse* blending factor, i.e., the higher alpha the less blending.

- *Vel-alpha influence* uses the "pixel-speed", i.e. the difference of the calculated 2D history-UV coordinates and the current sample UVs, to determine a pixel-specific alpha. The idea behind this is that areas with fast movement need to weigh current frame samples stronger, while more quiet areas can better rely on the accumulated history.
- *Rejection alpha* defines the alpha value to use when history is rejected. (The default value of 1.0 means: use the current sample.)

**Other parameters**

- *Unjitter neightbourhood* along with *unjitter current sample* and *unjitter factor* are experimental settings used during development. Not useful in general.
- *Interpol* dictates the texture interpolation mode when accessing history samples. Possible values are bilinear, bicubic B-spline and bicubic Catmull-Rom interpolation. Shader-wise, bilinear is a simple texture lookup. The bicubic methods require multiple lookups (4 taps for B-spline, 9 for Catmull-Rom), but lead to much crispier results, especially when motion is involved (see Figures 2 and 9).

Figure 2: History sampling interpolation: Bilinear (left), Catmull-Rom (right). The camera was constantly moving left/right. Higher-order interpolations result in sharper anti-aliased images, especially when motion is involved.

- *Sharpening* applies a sharpening post-processing step. This can help to alleviate the subtle blurriness resulting from TAA in some areas. Two implementations are available: a simple, 5-tap gradient-based sharpener, and FidelityFX Contrast Adaptive Sharpening (CAS).
- *Dynamic antighosting* is an experimental setting to help with ghosting artifacts, inspired by Unreal Engine 4. The idea is to reject the history if there is no movement in a 5-tap neighborhood currently, but there was movement at the current pixel in the previous frame. This works reasonably well for locally restricted movers, but not for overall movement (i.e., camera motion), therefore the implementation is restricted to act on dynamic objects only.
- *Reduce blend* is an anti-flickering strategy, described by Karis [2]. It takes place after color clipping/clamping has occurred and considers the (luminance) distance of the current pixel to the clamping box. The blending factor is reduced (i.e., alpha is increased) the closer the pixel is to being clamped.
- *Noise* is a purely experimental setting, which adds noise to the output image. Not useful in general.

# 3 Adaptive Ray Tracing

In addition to techniques described by Yang et al. [4], adaptive ray tracing [3] was implemented, which identifies potentially problematic areas (see Figure 3 for an example) during the TAA shader pass: the screen-space derivatives of surface normals, depth values, material identifiers and luminance are calculated and added up to a weighted sum. Additionally, the pixels "ray tracing history information" is taken into account, i.e., whether it was marked for ray tracing in the previous frame(s). Pixels where this sum exceeds a user-settable threshold are marked for ray tracing. Finally out-of-history areas (like new areas at screen borders, when the camera is rotating) are identified and marked for anti-aliasing via FXAA. In *Anti-Aliasing Settings/Ray trace augment* the weights for the different derivatives as well as the overall threshold can be changed.

Using these values, a segmentation mask image, as shown in Figure 4, is generated. After the TAA pass has finished, the mask image is used to generate the missing data through ray tracing, only for those pixels which have been marked. Pixels marked as out-of-history are instead anti-aliased using FXAA. Additional details about this technique are given in [3].

One problem with ray tracing—when compared to the rasterizer—is that there is no automatic mip level selection for sampling textures. If ray tracing would always sample textures at the lowest mip level[5], this would lead to visible breaks between ray traced and rasterized parts of the image. One possibility to work around this is to always use the lowest mip level during rasterization too. This works, but effectively disables mipmapping, which leads to new problems like texture-aliasing (to experiment with this mode, disable *RT approximate LOD* and enable *always use lod 0* in *Info & Settings/Rendering*).

As a workaround, the application tries to calculate an appropriate mip level when ray tracing. When a ray-triangle intersection is processed, two addition (hypothetical) intersection points with the same triangle are calculated[6]: one with a ray that passes through the screen plane exactly the size of one pixel to the right of the original ray, and one that passes exactly the size of one pixel above. From the texture coordinates of these additional points their screen-space derivative can be obtained, and thus the correct mip level can be calculated. Note that while this mip calculation is exact for direct ray hits, it would not hold for secondary rays (reflections). This is not a problem in this application, since ray traced reflections are not used here, but if this is required, a more sophisticated scheme has to be used. Additional information about this topic is described by Akenine-Möller et al. [1].

The current implementation uses ray tracing pipelines, i.e., separate ray-generation-, hit- and miss-shaders. This design decision follows the approach described in [3], where ray tracing happens as a separate step after rendering and TAA. However, it would also be conceivable to integrate ray tracing directly into the TAA shader using ray queries.

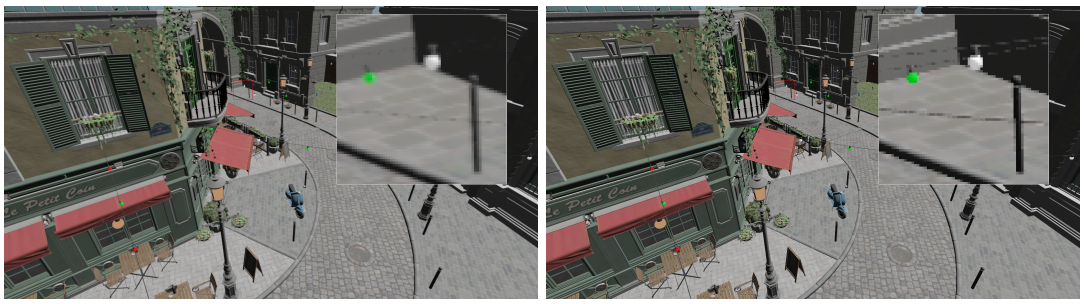

Figure 3: Very fine (sub-pixel) structures, like these wires, tend to vanish with TAA, especially when the camera is moving. Adaptive ray tracing can detect and fix such situations. Left: without, right: with adaptive ray tracing.

[5] Lowest mip level means sampling from the highest texture resolution.

[6] It is not required for the hypothetical intersection points to actually fall inside the area of the triangle - only the plane it spawns matters.

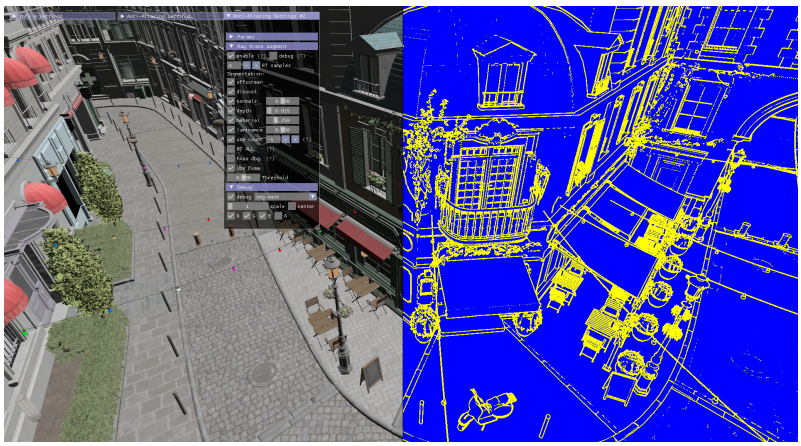

Figure 4: A mask image identifying problematic areas (yellow lines in above image) is generated during the TAA pass. For each of those problematic pixels, a certain number of additional samples (configurable via *Ray trace augment/RT samples*) are generated in a subsequent ray tracing pass. Out-of-history areas (not shown here) are antialiased using FXAA.

# 4 Temporal Upsampling

Basic support for temporal upsampling has been included in the project. It needs to be explicitly enabled via the command line parameter `-upsample <factor>`. This causes the rendering to be performed on a smaller buffer, so that the both the width and height of the output image are `<factor>` times larger than the rendered image. Experiments showed that the TAA parameters need to be tuned quite differently compared to using "plain TAA"; e.g., color clamping or clipping can be detrimental to the upsampled image quality and/or lead to flickering. It was also found that a larger alpha value (e.g., 0.2–0.3) and a more extensive jittering pattern, like Halton x16, helps. An example is shown in Figure 5. There is also an option to experiment with upsampling still images instead of a rendered 3D scene, which can be enabled in the main settings window (*Debug/image*).

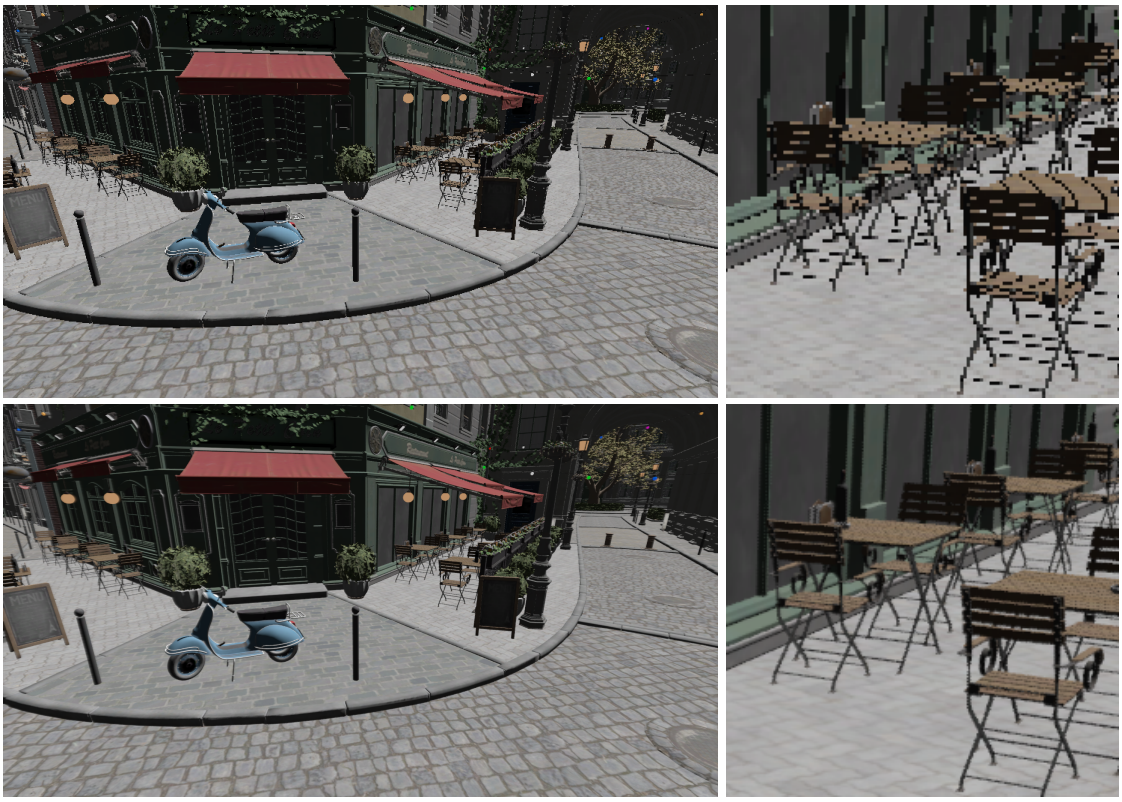

Figure 5: Temporal upsampling. The scene is rendered with only half the display resolution (top row) and then upsampled (bottom row).

# 5 Split-Screen and Magnification

To compare different TAA settings side-by-side, a split screen display can be enabled in *Anti-Aliasing Settings/Split screen*. When active, a second parameter window labeled *Anti-Aliasing Settings #2* for controlling the parameters on the right side of the splitter is shown. Parameters can be copied from the left to the right side or vice versa with the *1→2* and *1←2* buttons, or flipped with the *flip* button. The splitter can be dragged with the mouse.

To zoom in on areas of interest, a magnification box is available via *Anti-Aliasing Settings/Post-processing/zoom*. Both the source area and its magnified content can be freely dragged and resized on the screen; when the `Shift` key is held down while resizing, a square layout is enforced.

# 6 Saving and Loading

When experimenting on a specific area of interest, or a certain combination of parameters, it can be very tedious to re-enter all values when the application is started anew. Therefore the current settings, along with the camera position, can be saved to (and loaded from) a file—also useful for transferring settings to a different PC. In addition, some descriptive text can be saved along with the settings. The functionality is accessed with the buttons *Load*, *Save* and *Notepad* at the bottom of the main settings window.

Note: Besides camera orientation, the saved file includes all interactive parameters. But it does **not** include the command line parameters; the 3D scene choice itself is not saved.

# 7 Path Editor

To experiment with automatic camera movement other than simple repeating motion (like strafing, bobbing, rotation, which can be set in *Info & Settings/Camera*) an interactive path editor is included in the application, as shown in Figure 6. This allows to define an interpolated curve which the camera can follow. The editor is accessed via the *edit* button in *Info & Settings/Camera*.

The type of interpolation curve (Bezier, quadratic B-spline, cubic B-spline or Catmull-Rom) and global parameters like fly-through duration, constant speed, look-along vs. free look can be selected. Path control points can be entered manually or manipulated directly on-screen by dragging them with the mouse. When dragging, movement is restricted to the horizontal plane; to change the vertical position of a control point the `Shift` key can be held down while dragging.

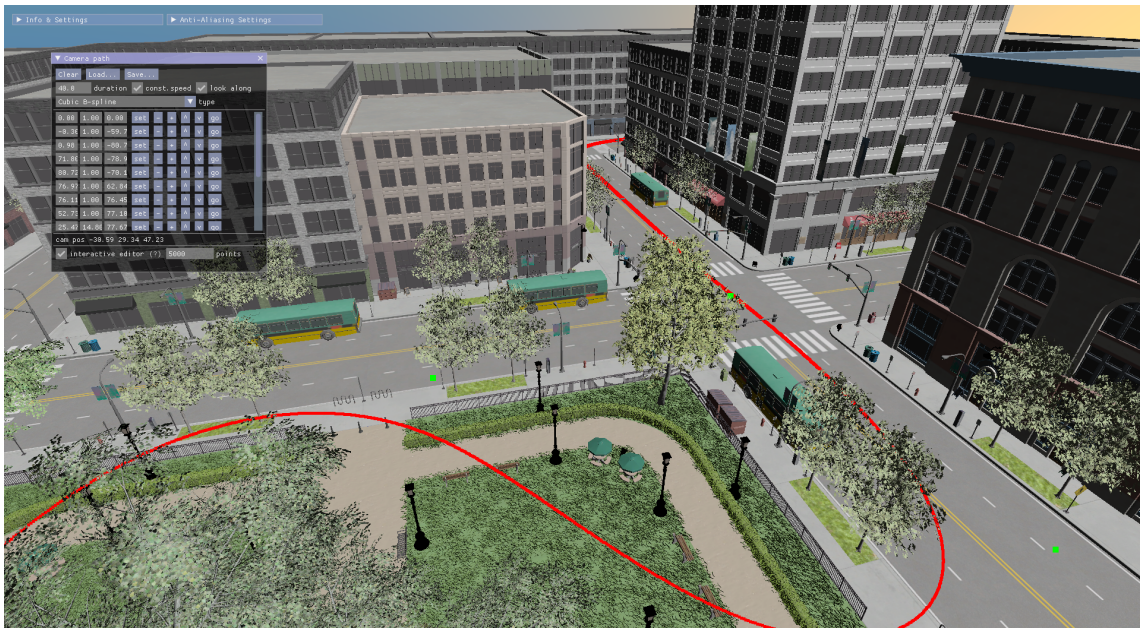

Figure 6: The interactive path editor can be used to design camera fly-through paths.

# 8 Discussion

Trying to find a one-fits-all parameter set that is suitable for every scene is a seemingly impossible task. Most of the techniques address very specific problems, and in general they work quite well where those are present. However, chances are that a technique or setting that fixes such a certain very specific problem impairs different areas of the same scene which are not affected by that certain problem.

Some of the scene characteristics that hugely affect technique/parameter choice are:

- Are surfaces very rough with a lot of apparent color-change when viewed close up, or is the geometry rather mostly smooth?
- Is there a lot of color variety due to strong specular reflections?
- Do very fine-grained structures (like fences, thin wires) exist in the scene?
- Does the scene use many small irregular shapes in a close area (like leaves of a tree, or grass blades)?
- Is there a lot of moving geometry?

In general one can not answer such questions unambiguously for the whole scene, so it is often necessary to compromise, i.e., choose techniques and parameters that work well for most parts of the scene without compromising other parts too much. This is, unfortunately, mostly a trial-and-error process.

If it is possible to logically split the scene into areas requiring different TAA settings, this should be exploited: For instance, one could switch to a different parameter-set when the camera moves from outside scenery to the inside of a building. This is quite easy to implement when there is a hard scene-switch. But even if the scene is widely open it might prove useful to define different settings for a couple of areas.

#### Recommended Settings

Settings that were found to work well for different scenarios are:

*Wide open areas:*

- Clamping or clipping are essential. Far away geometry will move a large screen-space distance when rotating the camera and thus is prone to motion blur. Fine-tune the clamping/clipping parameters.
- Depth culling seems work well here and reduces blurring a bit (whereas in other scenarios depth culling often leads to flickering).
- Sharpening helps to improve contrast.

*Very fine structures:*

- Clamping or clipping can be problematic and may have to be turned off completely.
- Adaptive Ray Tracing is recommended to preserve details.

*Rough areas with lot of color change:*

- Variance clipping in YCoCg space seems to work better than clamping.
- Adaptive ray tracing can improve the visual result. However, it may be costly, because large areas will be affected due to the high color variance.
- Sharpening should rather be avoided, because it can lead to slightly blurred patches when moving the camera.

*Foliage, grass blades, fine transparent geometry:*

- The main problem in this scenario is motion blur. Clamping/clipping is essential. Shaped neighborhood clipping seems to work better than variance clipping for these structures.
- Adaptive Ray Tracing helps a lot. But again, it may be costly performance-wise.
- Sharpening is recommended. Without it, foliage tends to look bland.

### Example: City-Like Scene

A city-like scene could contain park areas with lots of trees, grass and bushes, as well as straight street canyons. In that case one could either switch parameters when moving across pre-defined border-locations, or depending upon which scene-areas take up most of the screen-space. Ideally one would smoothly blend parameters when switching sets, so there is no discernible break in perception.

### Example: Fine Structures

There will likely still remain some pathological areas in most scenes that are very difficult to address, such as the perspective foreshortening of the fence as shown in Figure 7. The fine fence structure leads to very different images when the projection matrix is jittered. When either color clipping or clamping is active, the result is an annoying flickering in that area. In that case no combination of techniques could be found to counter that problem, except to disable all color- or luminance-based history correction.

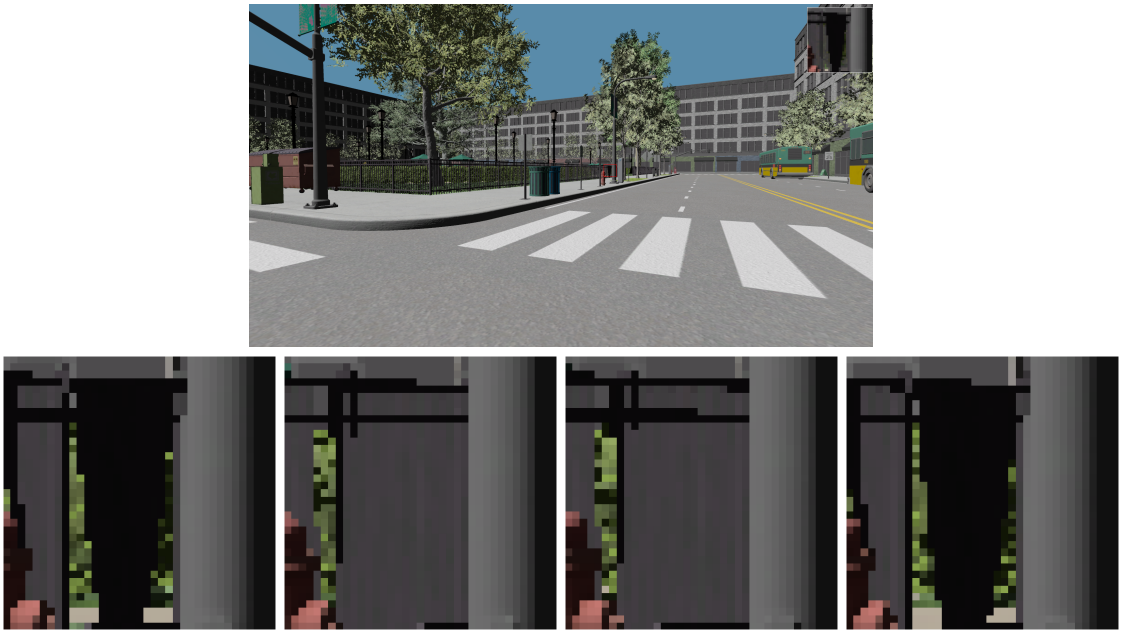

Figure 7: The slight shifting of the projection matrix due to TAA jittering can lead to problematic areas with massive color-changes. When using either color clipping or clamping this results in disturbing flickering. The bottom row shows the raw images generated for subsequent jittering steps.

### Example: Animated Model on Rough Terrain

Contrary to the above, Figure 8 shows a situation where clipping is badly needed. This example also demonstrates how settings that usually work quite well, like color clipping in RGB space, can dramatically fail under certain circumstances. Here, the street roughness was intentionally exaggerated to simulate uneven terrain. This introduces a high variability of colors in the background of the animated object, which is detrimental to clipping and results in heavy ghosting. Performing the clipping in the YCoCg color space instead of RGB improves the situation noticeably, but a satisfactory ghosting suppression could only be achieved by using variance clipping in addition.

Figure 8: Due to the high background frequencies (exaggeratedly rough floor), more sophisticated clipping methods are necessary to avoid ghosting. From left to right: clipping in RGB space, clipping in YCoCg space, variance clipping in YCoCg space.

### Higher-Order Interpolation Scheme

TAA tends to add a slight blurriness to the resulting image whenever camera movement is involved. One method that proved very useful in general is to use a higher-order interpolation scheme when sampling from the TAA history buffer, as demonstrated in Figures 2 and 9. This alleviates the washed-out look perceivably. Alternatively (or also in addition) a sharpening post-processing pass can be used. In comparison, sharpening turned out to be less effective at counteracting the smoothing-problem though.

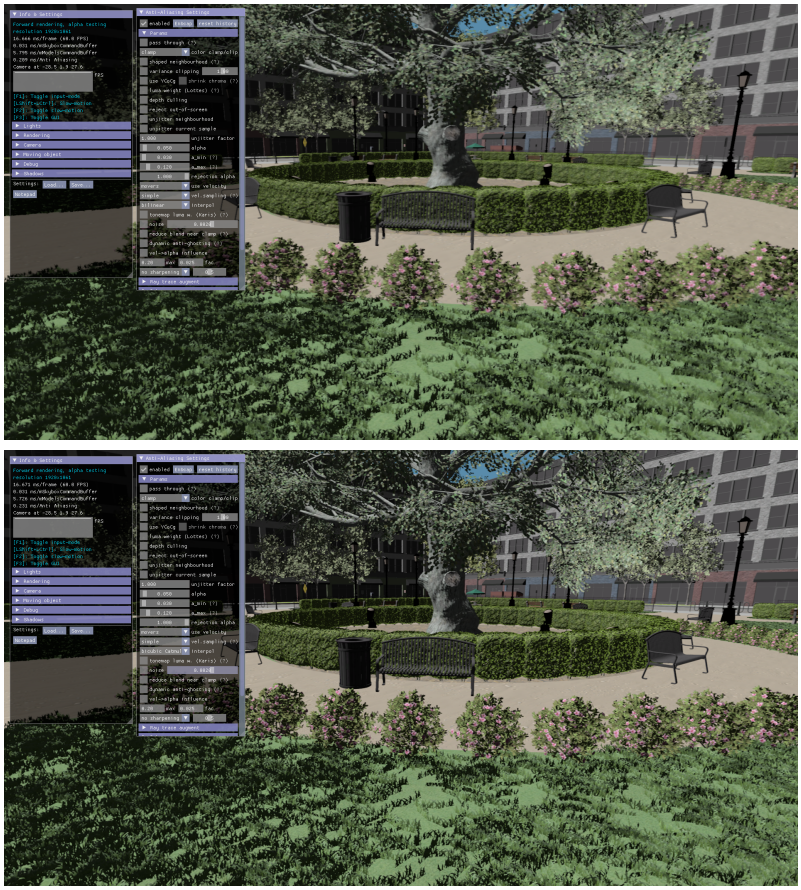

Figure 9: Using Catmull-Rom interpolation (bottom) instead of bilinear texture lookups (top) when sampling from the history buffer results in sharper images during camera movement. In the shown image this is most noticeable at the blossoms of the rose bushes, the details of the park bench and the leaves of the tree.

## References


[1] Tomas Akenine-Möller, Jim K. Nilsson, Magnus Andersson, Colin Barré-Brisebois, Robert M. Toth, and Tero Karras. Texture level of detail strategies for real-time ray tracing. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, pages 321–345, Jan 2019.

[2] Brian Karis. High Quality Temporal Supersampling. *ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games*, 2014.

[3] Adam Marrs, Josef Spjut, Holger Gruen, Rahul Sathe, and Morgan McGuire. *Improving Temporal Antialiasing with Adaptive Ray Tracing*. Apress, Berkeley, CA, USA, 2019.

[4] Lei Yang, Shiqiu Liu, and Marco Salvi. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum*, 2020.

# A Appendix: Command Line Parameter Summary

Usage: taa.exe [optional parameters] [orca scene file path]

Parameters:

| | |
|---|---|
| -w ⟨width⟩ | set window width |
| -h ⟨height⟩ | set window height |
| -small | use smaller default window size (1280x720) instead of (1920x1080) |
| -fullscreen | enable fullscreen mode |
| -upsample ⟨factor⟩ | upsampling factor (render framebuffer is ⟨factor⟩ times smaller than the window) |
| -sponza | ignore scene file path and load Sponza scene |
| -test | ignore scene file path and load Test scene |
| -device ⟨hint⟩ | device hint for GPU selection (e.g., -device INTEL or -device RTX) |
| -novalidation | disable validation layers (in debug builds) |
| -validation | enable validation layers (in release builds) |
| -gpuassisted | enable GPU-Assisted validation extension |
| -bestpractices | enable best practices validation extension |
| -blend | use alpha blending for transparent parts |
| -noblend | use alpha testing for transparent parts |
| -nomip | disable mip-map generation for loaded textures |
| -vsync | enable vsync (cap frames/sec to monitor refresh rate) |
| -hidewindow | hide render window while scene loading is in progress |
| -capture ⟨numFrames⟩ | capture the first ⟨numFrames⟩ with RenderDoc (only when started FROM RenderDoc) |
| – | terminate argument list, everything after is considered the scene path |

# B Appendix: User Interface Summary

## Window Info & Settings

### Lights

| | |
|---|---|
| max point lights | Maximum number of point lights to render |
| max spot lights | Maximum number of spot lights to render |
| dir light | Color and direction vector of the directional light |
| dir boost | Amplification factor for directional light |
| amb boost | Color and amplification factor for ambient light |
| Lighting | Switch between Blinn-Phong lighting or several debug modes |

### Rendering

| | |
|---|---|
| Cap framerate | Cap the frame rate to a specific number of FPS |
| Ray trace whole scene | Use ray tracing for the whole scene |
| RT samples | Number of samples per pixel when ray tracing |
| RT debug sparse tracing | Show areas to be sparsely traced instead of actually ray tracing them |
| RT approximate Lod | Use texture LOD approximation when ray tracing |
| aniso | Anisotropy factor when using texture LOD approximation |
| alpha thresh. | Consider anything with less alpha completely invisible (even if alpha blending is enabled) |
| alpha blending | Use alpha-blending instead of alpha-testing for transparent textures |
| lod bias | Manual LOD bias for texture sampling |
| taa only | Use the LOD bias only when TAA is enabled |
| always use lod 0 | Always sample textures with LOD level 0 (to match ray tracing without LOD approximation) |
| use lod 0 for alpha test | Use LOD level 0 for alpha testing |
| normal mapping | Controls the amount of normal mapping |
| Re-record commands | For debugging only |

**Camera**

| | |
|---|---|
| move | Toggle all automatic camera movement, set movement units |
| set&cap | Enable all automatic camera movement and take a capture in RenderDoc |
| rotation | Toggle camera rotation, set rate for vertical/horizontal axis |
| bobbing | Toggle camera bobbing |
| strafing | Toggle strafing, set strafing speed and distance |
| save cam | Remember current camera position and rotation |
| restore cam | Restore previously stored camera position and rotation |
| print cam | Print camera information to console |
| preset | Pick one of the predefined presets for camera position/rotation |
| follow path | Enable camera path traversal |
| reset | Restart camera path traversal |
| edit | Open the camera path editor |
| cycle | Toggle cyclic movement along the camera path |
| look along | Toggle free look/look along path when following camera path |
| detach | For debugging only |
| set to cam | For debugging only |

**Moving object**

| | |
|---|---|
| enable | Enable rendering of the moving object |
| type | Select type of moving object |
| start | Define start position |
| end | Define end position |
| rot ax/spd | Define rotation axis and rotation speed |
| speed | Define movement speed |
| conv | Convert speed settings between per-frame and per-second |
| repeat | Select cyclic or ping-pong movement between start and end position |
| cont rot | Toggle continuous (independent of position) rotation |
| reset | Reset movement |
| anim | Select animation frame (animated objects only) |
| auto | Toggle automatic animation (animated objects only) |
| speed | Set animation speed (animated objects only) |

**Debug**

| | |
|---|---|
| image | Show a 2D image instead of the 3D scene (for temporal upsampling) |
| bilinear sampling | Toggle between bilinear/nearest neighbor sampling for the 2D image |
| Regen.scene buffers | For debugging only |
| Cull view frustum | Toggle view frustum culling |
| capture | Take a capture with RenderDoc |
| frames | Number of frames to capture with the capture button |
| Show ImGui demo window | Show the ImGui demo window (for development use only) |

**Shadows**

| | |
|---|---|
| enable | Enable shadows |
| transp. | Toggle shadows of transparent objects |
| show shadowmap | For debugging only |
| show frustum | For debugging only |
| restrict to scene | For debugging only |
| num cascades | Set number of cascades for cascaded shadow mapping |
| Casc | Define cascade distances manually or automatically |
| manual bias | Set a manual bias term for shadow mapping |
| Depth bias | Define depth bias (constant, slope, clamp) parameters for each shadow cascade |

### Other

| | |
|---|---|
| Settings: Load/Save | Load or save the current settings to/from a file |
| Notepad | Open the notepad window to add remarks (text is saved along with settings) |

## Window Anti-Aliasing Settings

| | |
|---|---|
| enabled | Masterswitch to enable/disable TAA |
| En&cap | Enable TAA and immediately take a capture with RenderDoc |
| reset history | Reset the TAA history buffer |

### Params

| | |
|---|---|
| pass through | For debugging: just output the rendered scene unchanged |
| color clamp/clip | Enable color clamping or color clipping |
| shaped neighborhood | Toggle shaped neighborhood clipping: averages the min/max of 3x3 and 5-tap clipboxes |
| variance clipping | Toggle variance clipping and set its gamma-parameter |
| use YCoCg | Use the YCoCg color space for all calculations instead of RGB |
| shrink chroma | Reduces the chroma influence on the color clip box (only when using YCoCg) |
| luma weight (Lottes) | Dynamic luma weighting: adjusts alpha depending on the luma difference between history and current image; see also a_min and a_max |
| depth culling | Toggle depth culling, i.e. history rejection due to depth differences |
| reject out-of-screen | Reject samples that fall outside the dimensions of the history buffer |
| unjitter neighborhood | For debugging only |
| unjitter current sample | For debugging only |
| unjitter factor | For debugging only |
| alpha | Define the alpha parameter for TAA, i.e., the blend factor defining how fast previous history is faded out |
| a_min | Minimum alpha value for luma weight (Lottes) |
| a_max | Maximum alpha value for luma weight (Lottes) |
| rejection alpha | Alpha value to use for rejected samples (typically 1, i.e., use the current image value) |
| use velocity | Select when to use velocity vectors (not at all, for moving objects or for everything) |
| vel.sampling | Sampling strategy for velocity vectors: *simple* just samples velocity at the current fragment; *3x3 longest* takes the longest velocity vector in a 3x3 neighborhood; *3x3 closest* takes the velocity from the (depth-wise) closest fragment in a 3x3 neighborhood" |
| interpol | Select between *bilinear*, *bicubic b-Spline* or *bicubic Catmull-Rom* interpolation for history sampling |
| tonemap luma w. (Karis) | Toggle tone mapping approximation via luma |
| noise | Experimental; add random noise to the output image |
| reduce blend near clamp | Helps to reduce flicker: Reduce the blend factor when the history is close to being clamped |
| dynamic anti-ghosting | Reject history if there is a no movement in a 5-tap neighborhood and there was movement at the current pixel in the previous frame (inspired by Unreal Engine) |
| vel→alpha influence | Let the velocity vectors influence the alpha parameter (higher alpha for fast moving pixels); helps against ghosting |
| (Sharpener) | Choose between no sharpening, a simple, 5-tap gradient-based sharpener, and FidelityFX Contrast Adaptive Sharpening (CAS) |

**Ray trace augment**

| | |
|---|---|
| enable | Enable ray tracing augmentation |
| debug | Show areas to be sparsely traced instead of actually ray tracing them (same setting as in main window) |
| RT samples | Number of samples per pixel when ray tracing (same setting as in main window) |
| offscreen | Put offscreen pixels in the segmentation mask (for FXAA) |
| disoccl. | Put disocclusioned pixels in the segmentation mask (for ray tracing) |
| normals | Segmentation weight for the normal vector derivative |
| depth | Segmentation weight for the depth derivative |
| material | Segmentation weight for material differences |
| luminance | Segmentation weight for the luminance derivative |
| use count | Use a history counter: Pixels that are not marked for ray tracing in the current frame, but were marked in the previous $n$ frames will be ray traced |
| RT ALL | Debug setting - mark all pixels for ray tracing |
| Fxaa dbg. | Debug setting - a fixed sized border of the image is always marked for FXAA |
| Use Fxaa | Enable using FXAA for offscreen pixels |
| Threshold | If the weighted sum of the segmentation parameters exceeds this threshold, pixels are marked for ray tracing |

**Debug**

| | |
|---|---|
| debug | Enable (and choose type of) debug output |
| scale | Scale debug values before displaying them as RGB |
| center | Center debug values (multiply by 0.5 then add 0.5); applied after scaling |
| R,G,B,A | Toggle specific color channels for debug values |

**Split screen**

| | |
|---|---|
| split | Enable split screen display (and manually set the splitter position) |
| 1→2 | Copy parameter set 1 (left side) to 2 (right side) |
| 1←2 | Copy parameter set 2 (right side) to 1 (left side) |
| flip | Flip the two parameter sets |

**Jitter**

| | |
|---|---|
| sample pattern | Choose one of the predefined TAA jitter patterns |
| lock | For debugging: display a specific step of the jitter pattern |
| scale | For debugging: scale jitter offsets by the entered value |
| slowdown | Slower jittering; e.g., 3 means only advance the jitter step every third frame |
| rotate | Rotate the jitter pattern by the specified amount (in degrees) |
| Debug pattern | Allows to define a manual jitter pattern (active when pattern *debug* is chosen in *sample pattern*) |

**Postprocess**

| | |
|---|---|
| enable | Enable post processing (necessary for split screen or zooming) |
| zoom | Enable zooming |
| show box | Draw a red box around the zoom source area |
| rst | Reset zoom to default values |
| src | Manually enter position and size of the zoom source area |
| dst | Manually enter position and size of the zoom target area |

**Other**

| | |
|---|---|
| Reset history at any change | When enabled, the history is automatically reset when any parameter is changed |

## Window Camera path

| | |
|---|---|
| Clear | Clear the path |
| Load/Save | Load or save the path to/from a file |
| duration | Set the traversal duration in seconds |
| const. speed | Maintain constant speed during path traversal |
| look along | Look along the path when traversing |
| type | Type of the path interpolation: Bezier, quadratic B-spline, cubic B-spline or Catmull-Rom |
| (Control points) | Position of the control points can be edited here |
| set | Uses the current camera position for the control point |
| - and + | Removes the control point or inserts a new control point |
| ^ and v | Moves the current control point up or down in the list |
| interactive editor | Enables the interactive editor (shows path and allows to drag control points in the scene) |
| points | Number of points used for rendering the path |