Fun with Fixed Function

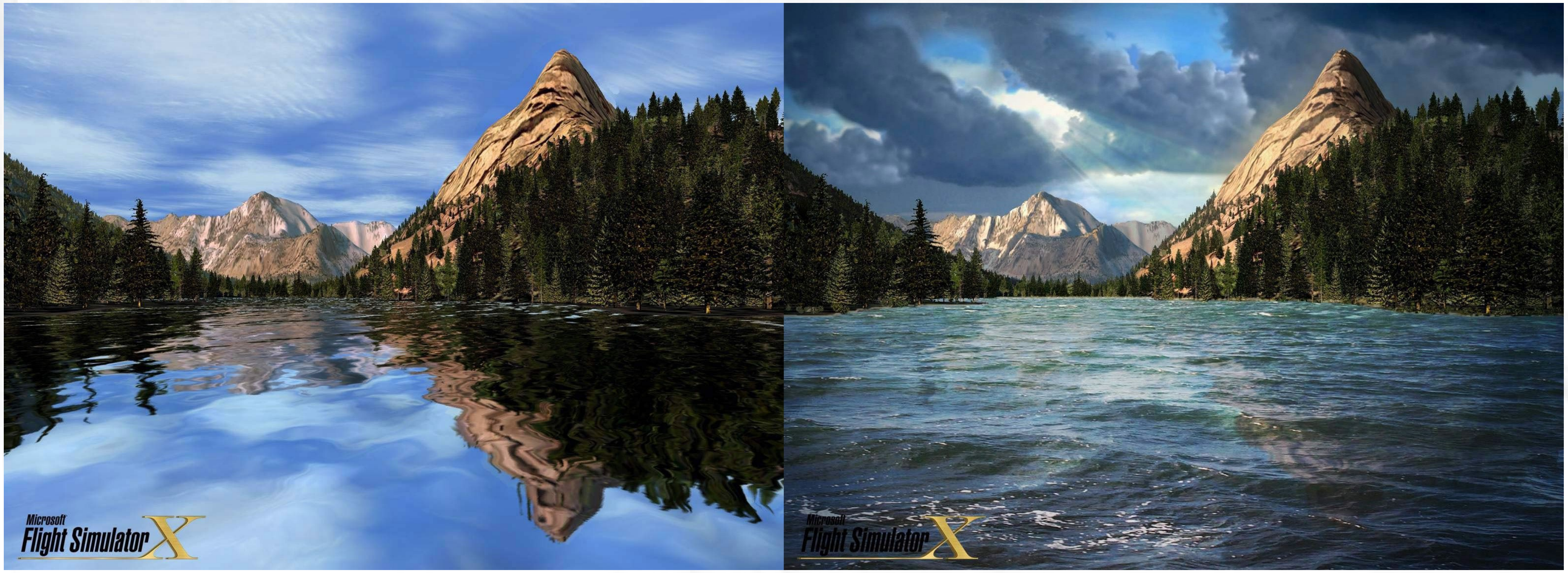
Tips for creating impressive visual effects on limited hardware

Caster

Inspiration and Faith

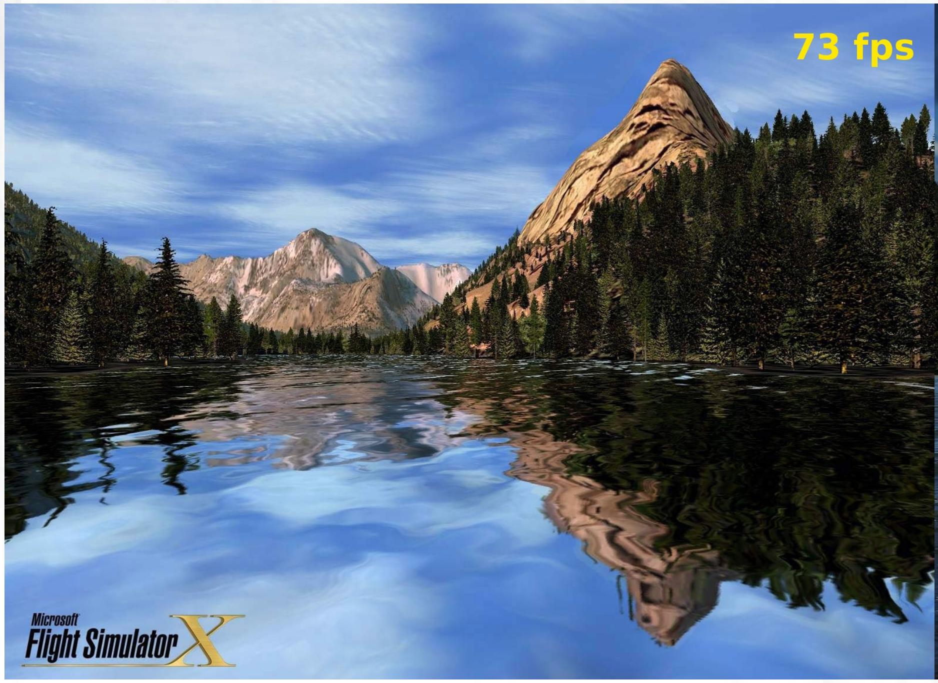
Comparison shots DirectX 9 and DirectX 10

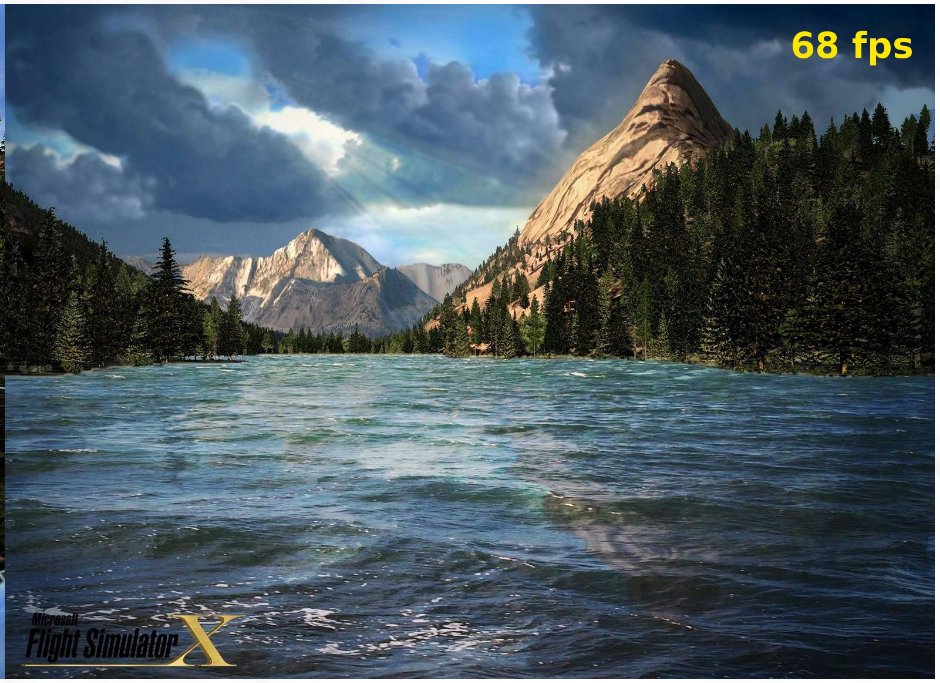
What is missing?



Comparison shots
DirectX 9 and DirectX 10

There we go!

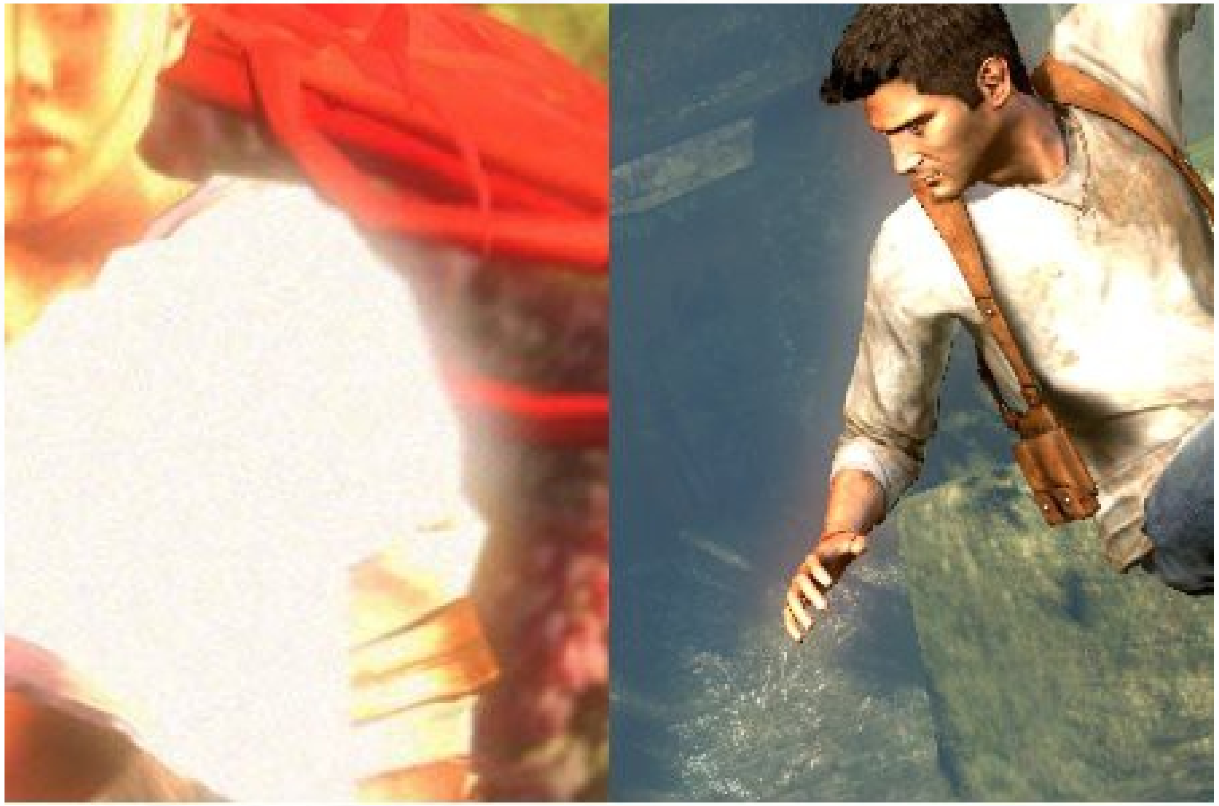




Graphics in Caster

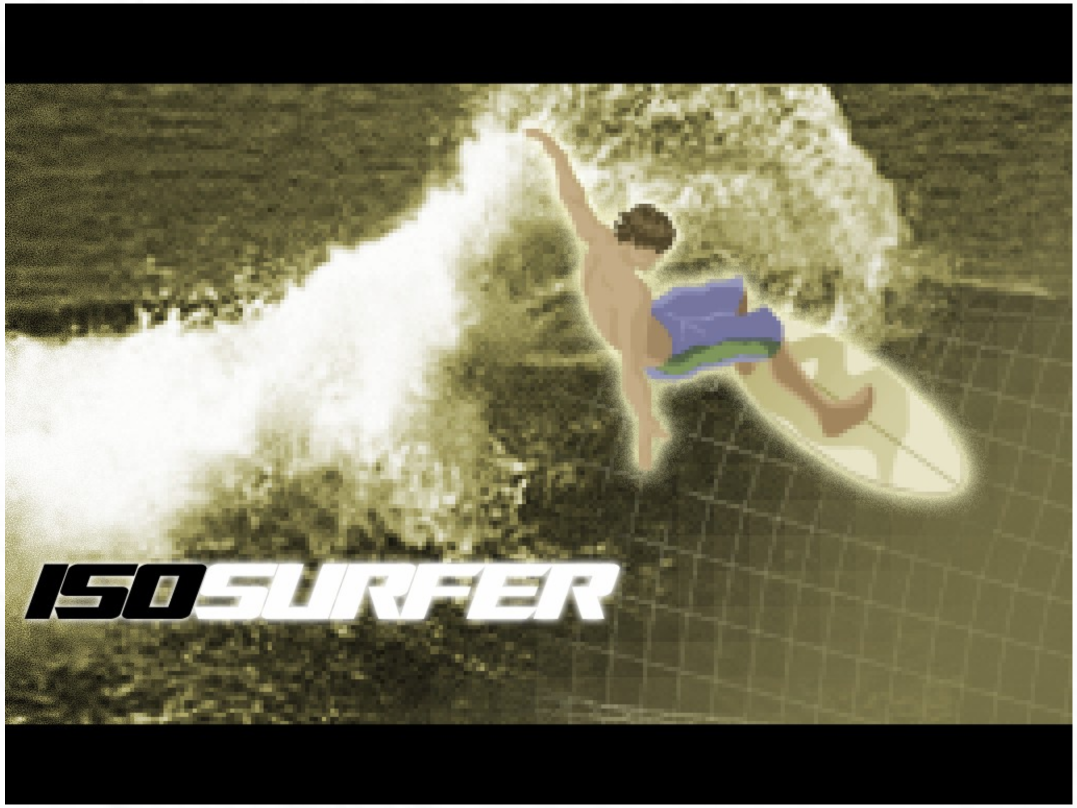
- Bloom Glow
- Motion Blur
- Depth of Field
- Screen Warp
- Temporal Anti Aliasing
- Water
- Lava
- Fog Effects
- Shadows
- Speed trails
- Toon Shading
- Terrain Deformation and Effects
- Particle Effects
- Full Screen Effects

Bloom and Glow

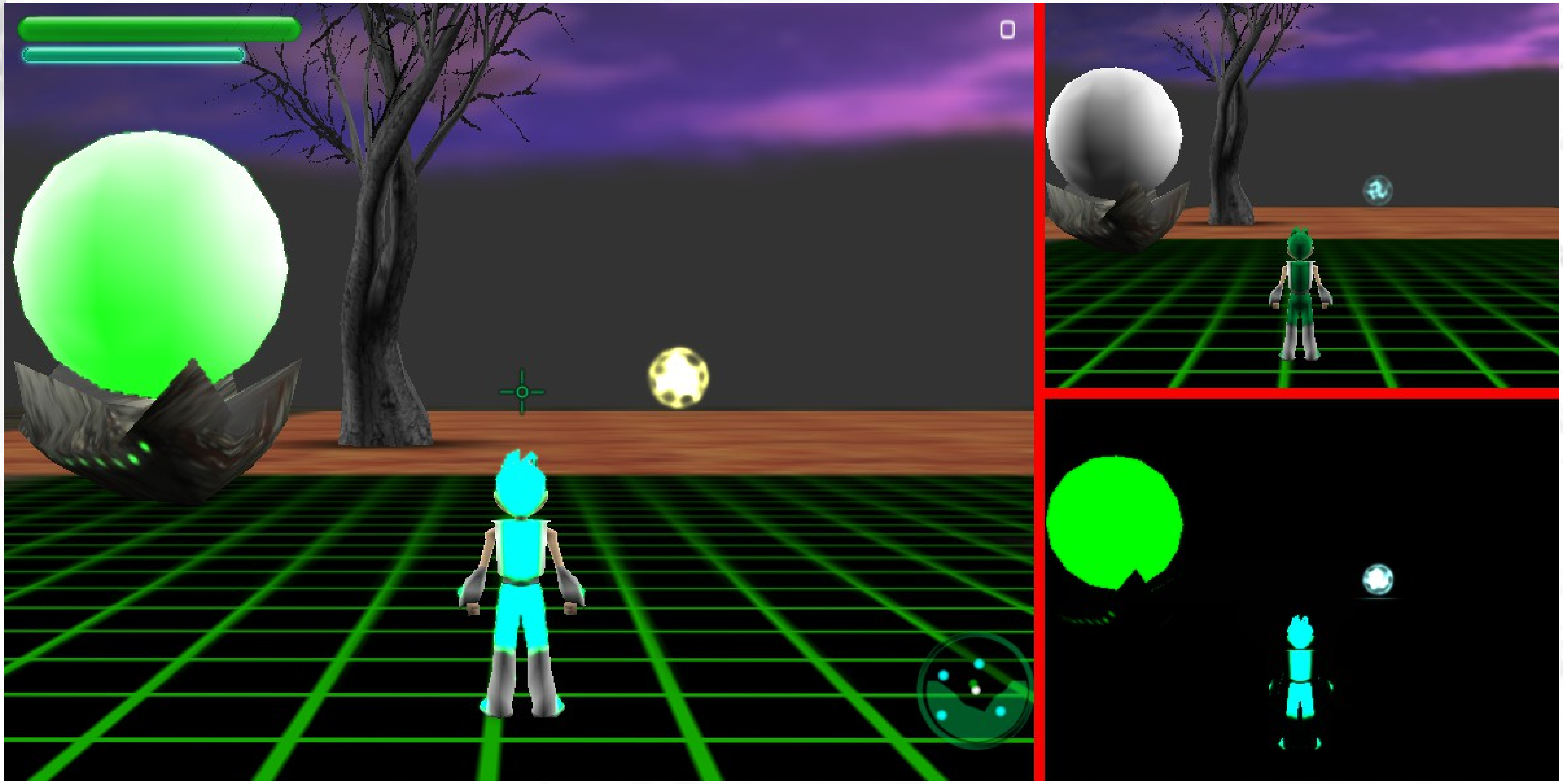


Do not want

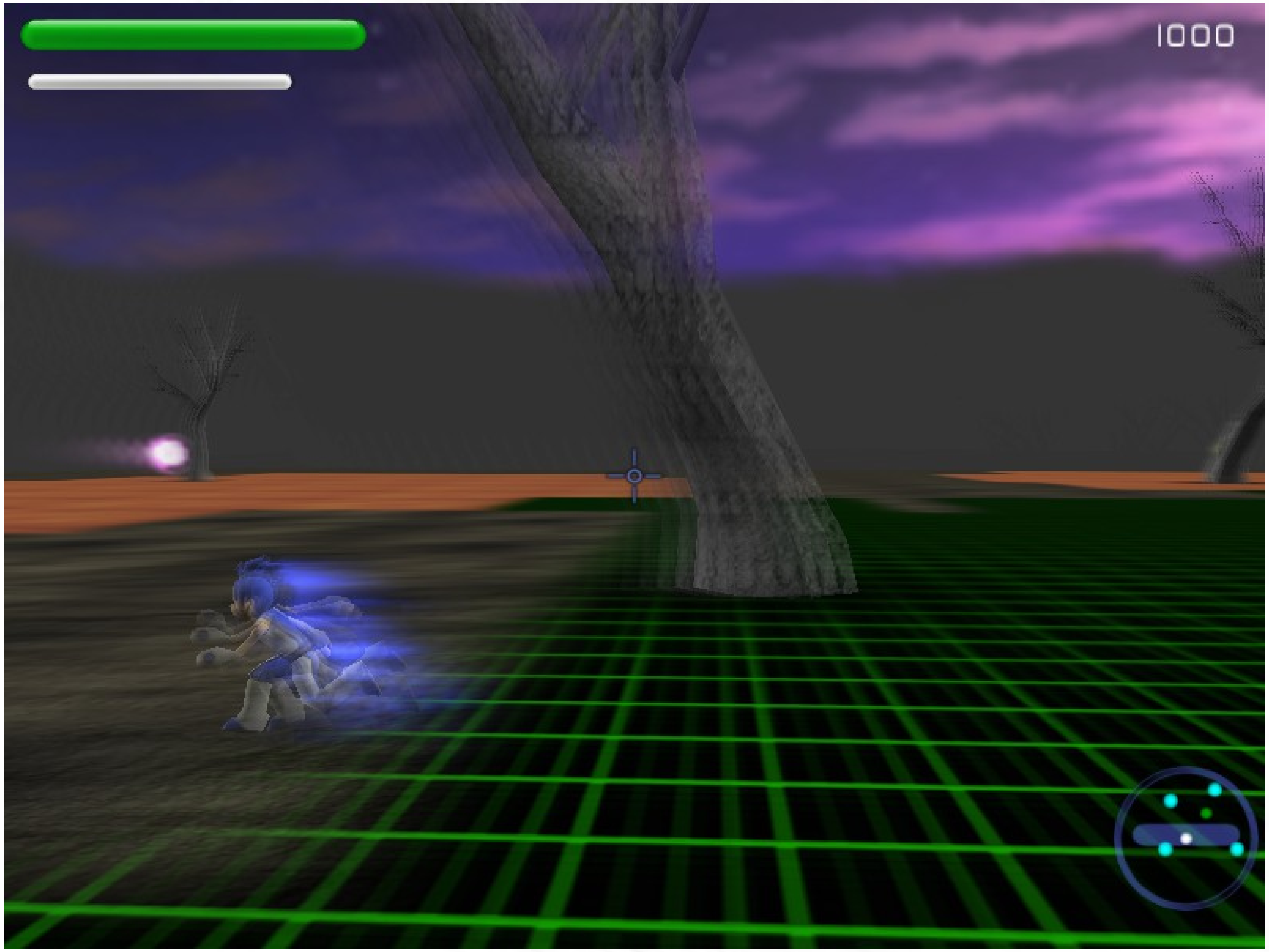
Bloom and Glow



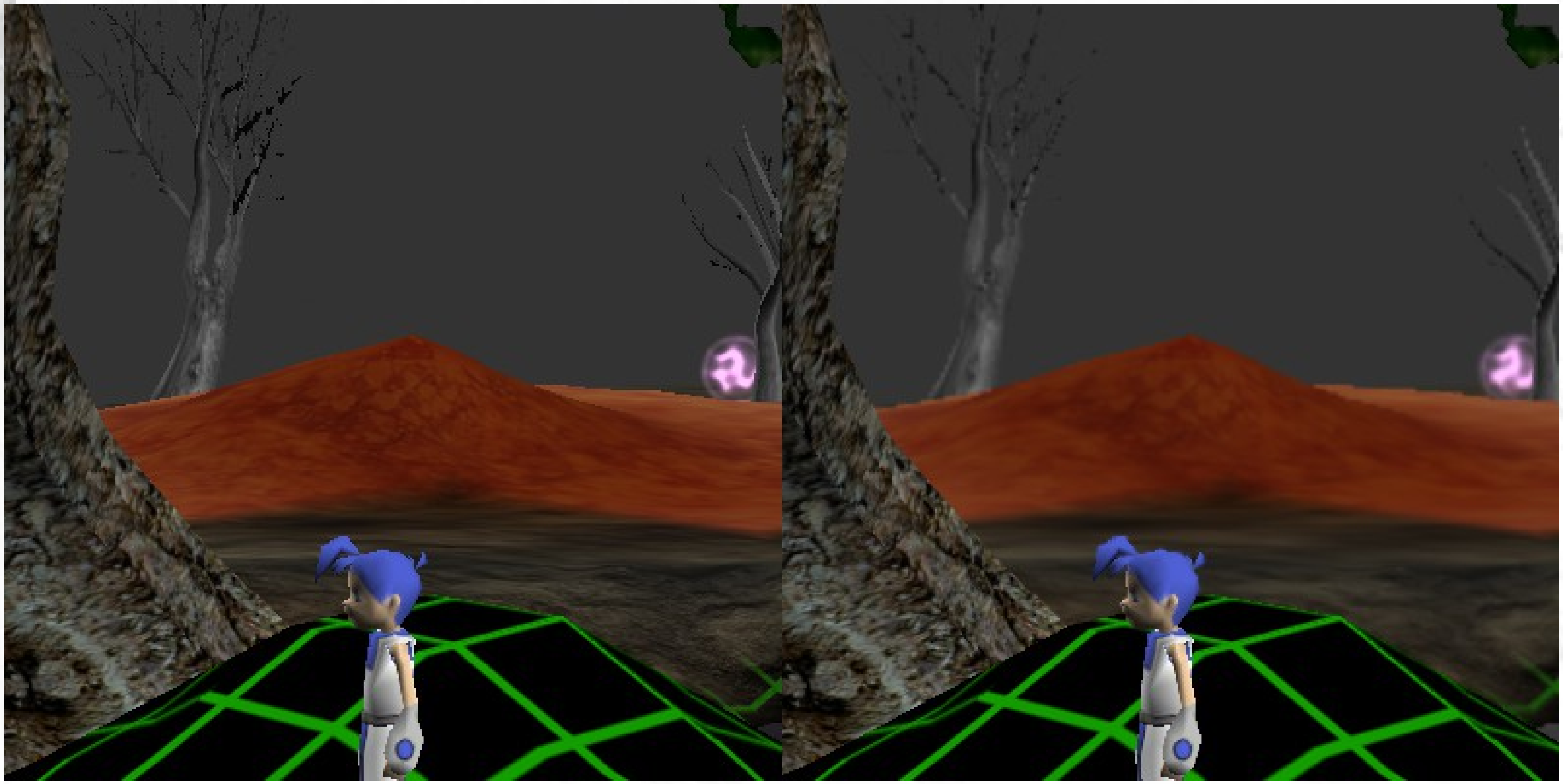
Bloom and Glow



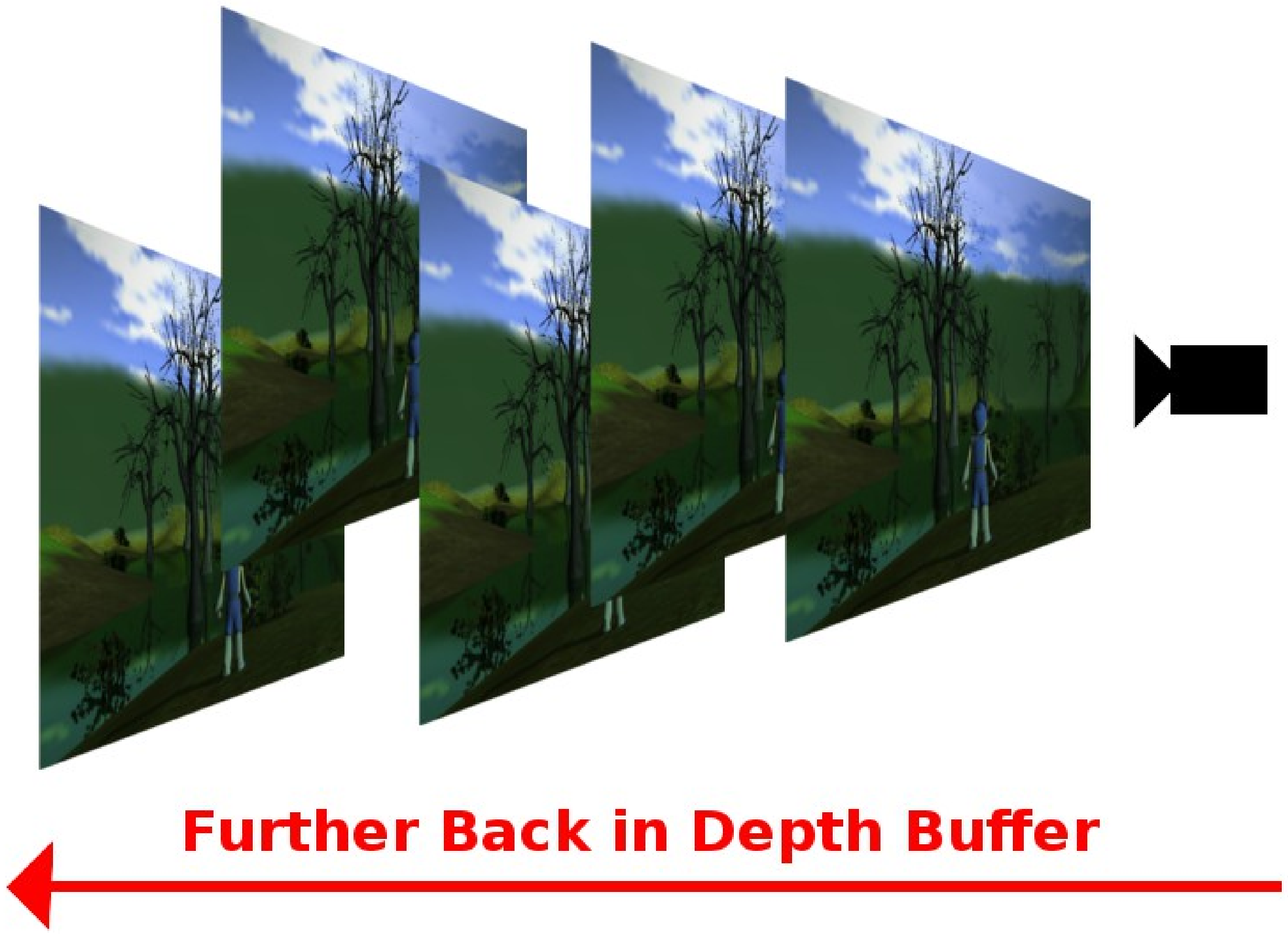
Motion Blur



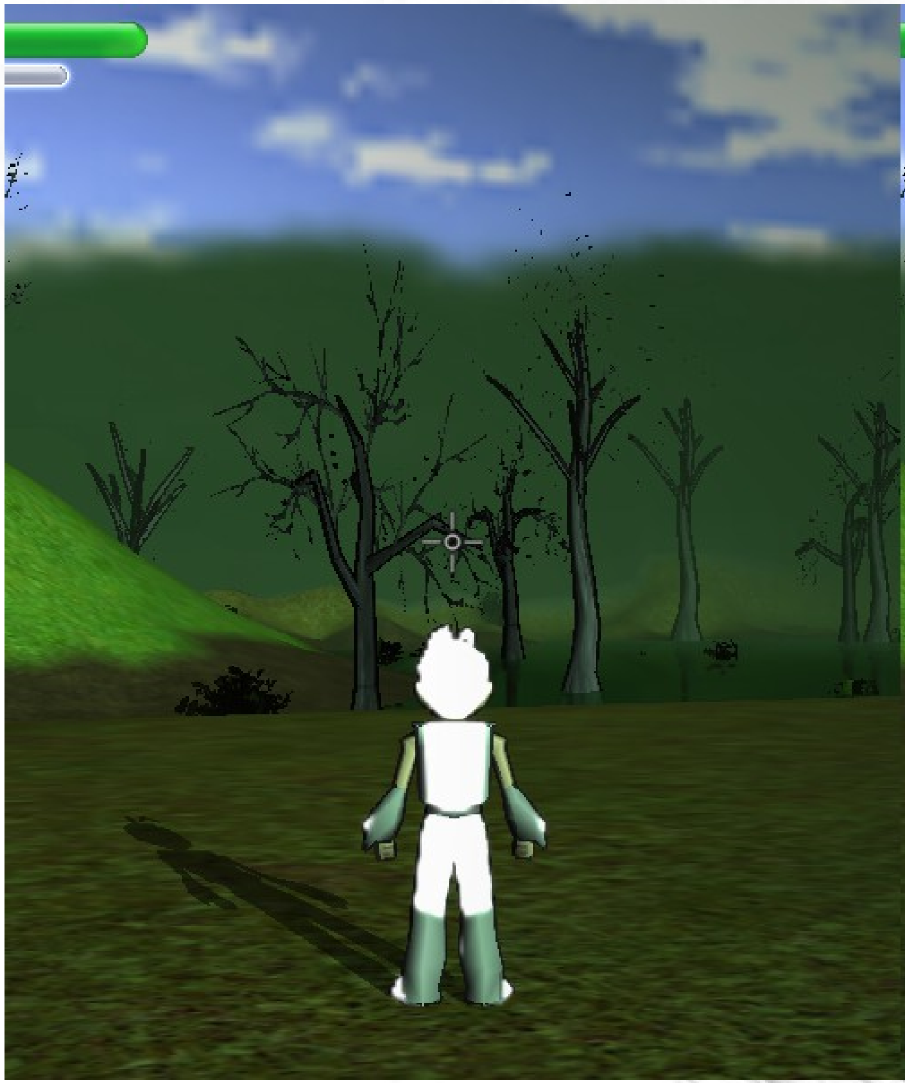
Depth of Field

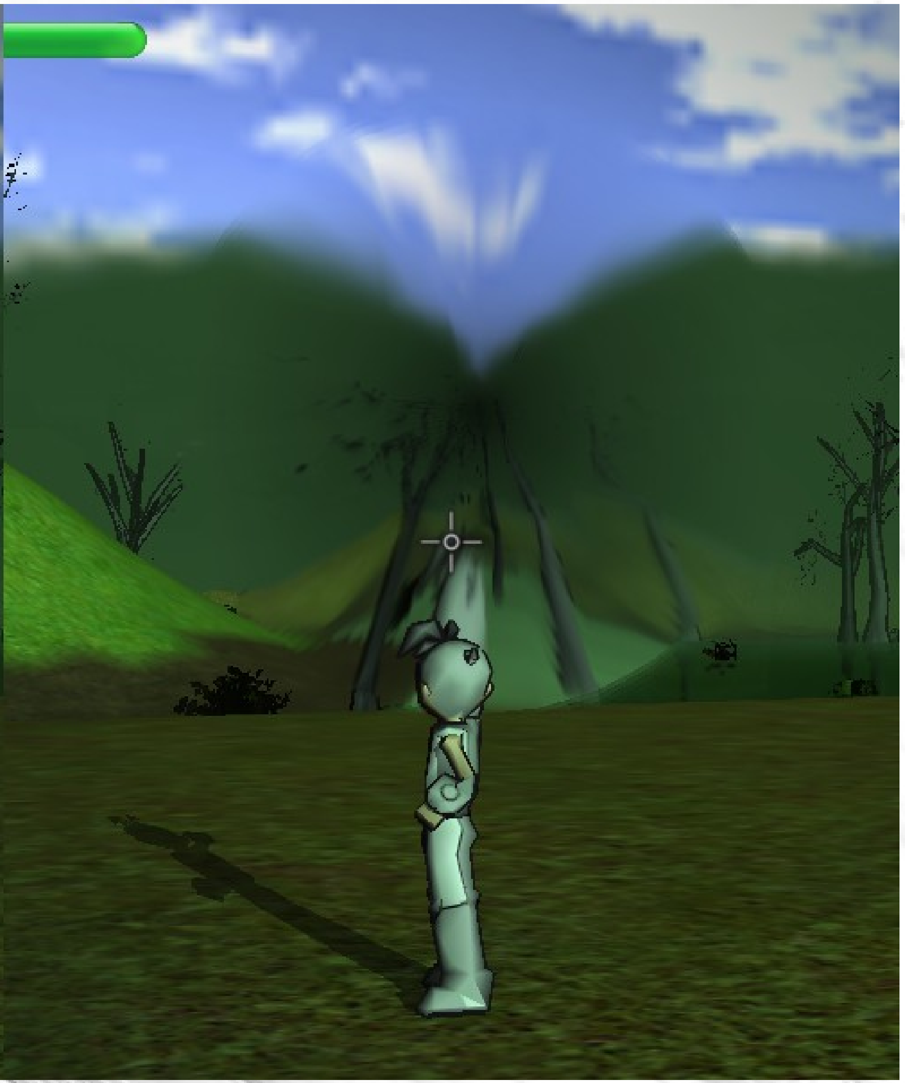


Depth of Field

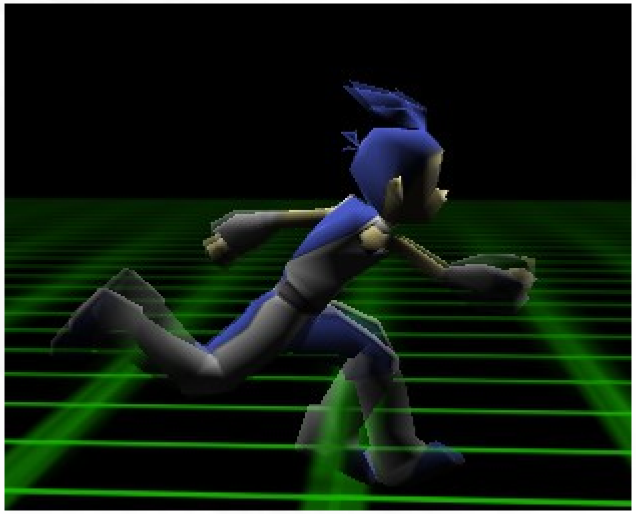


Screen Warp

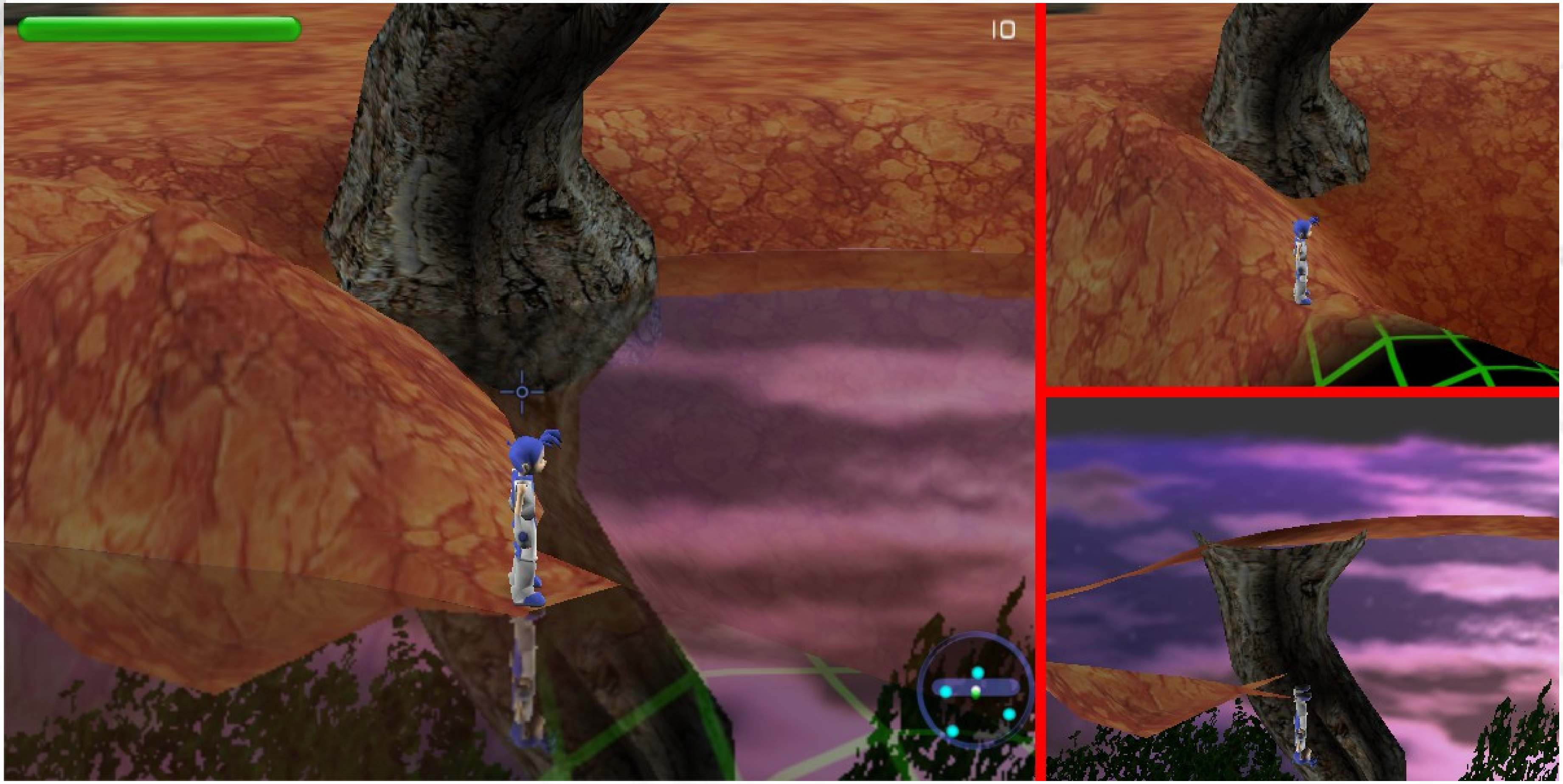




Temporal Anti Aliasing



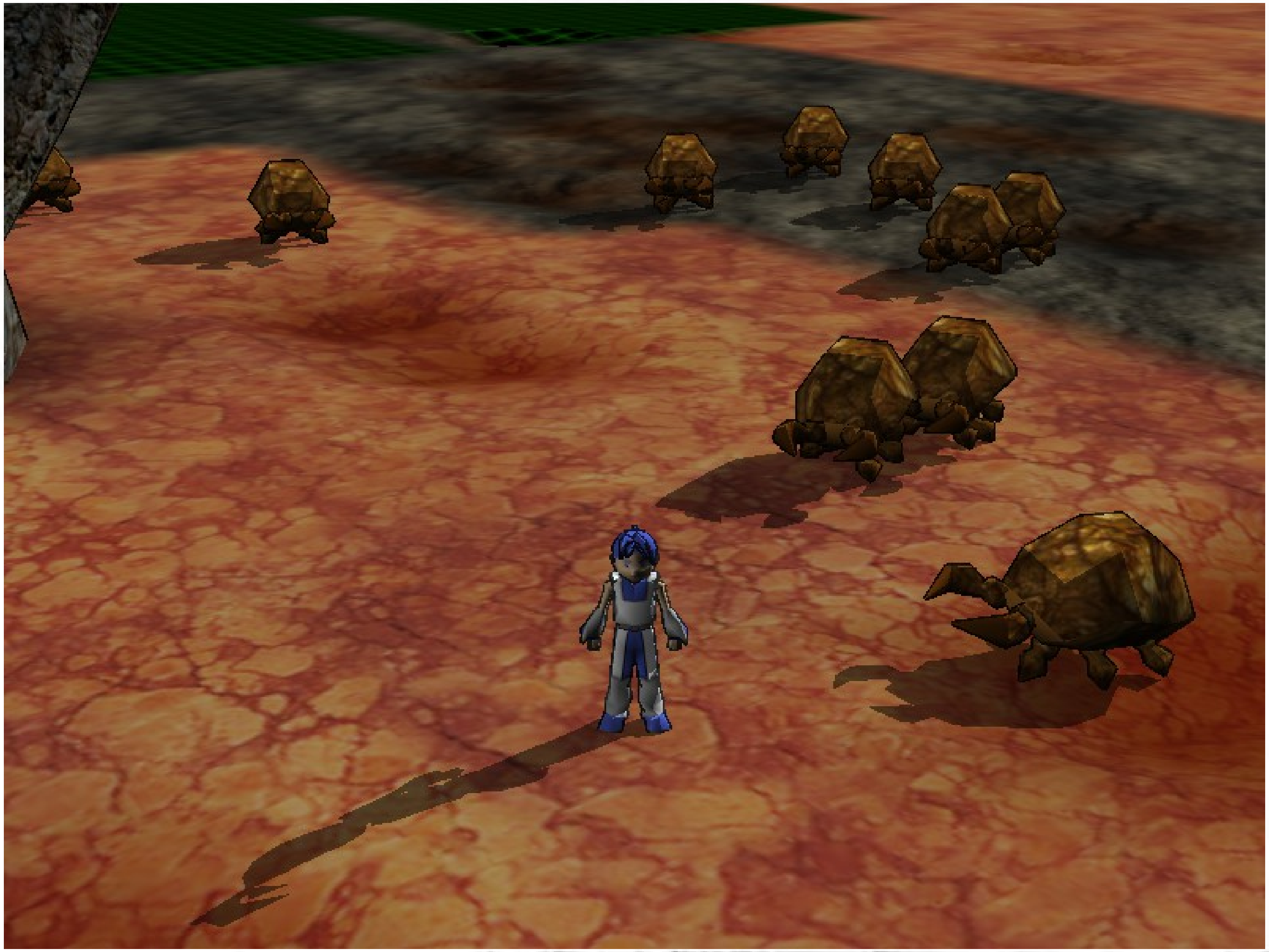
Water



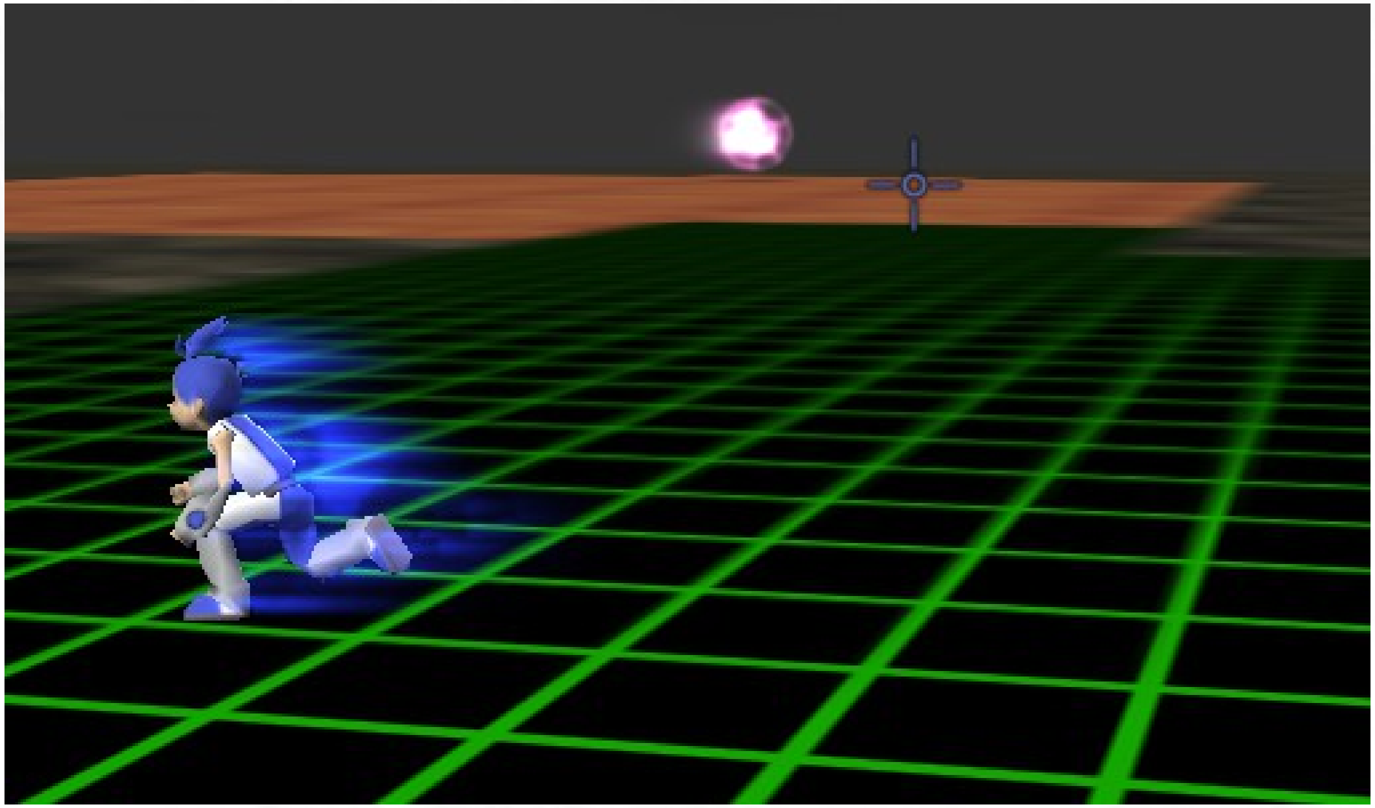
Lava

Fog Effects

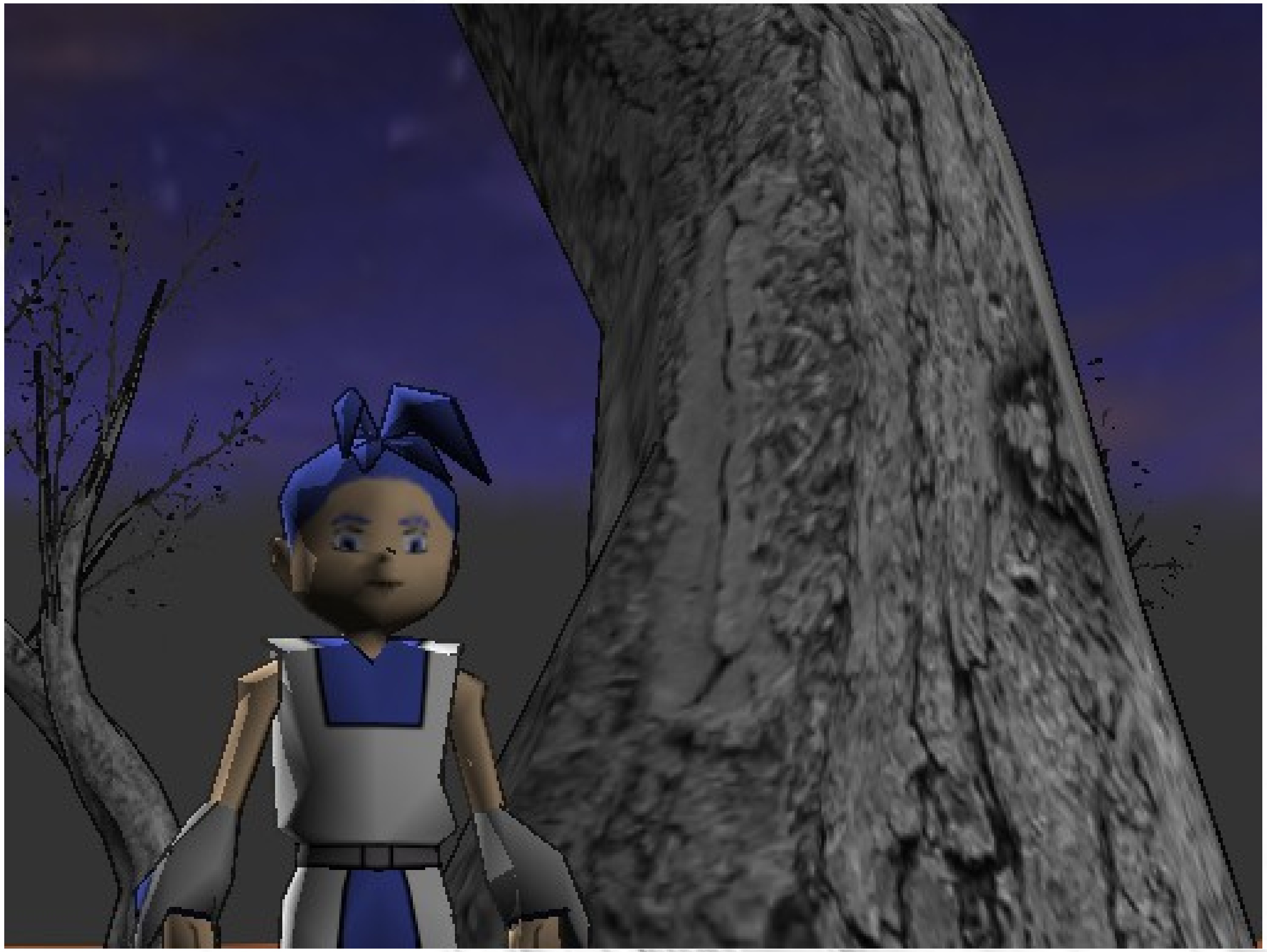
Shadows



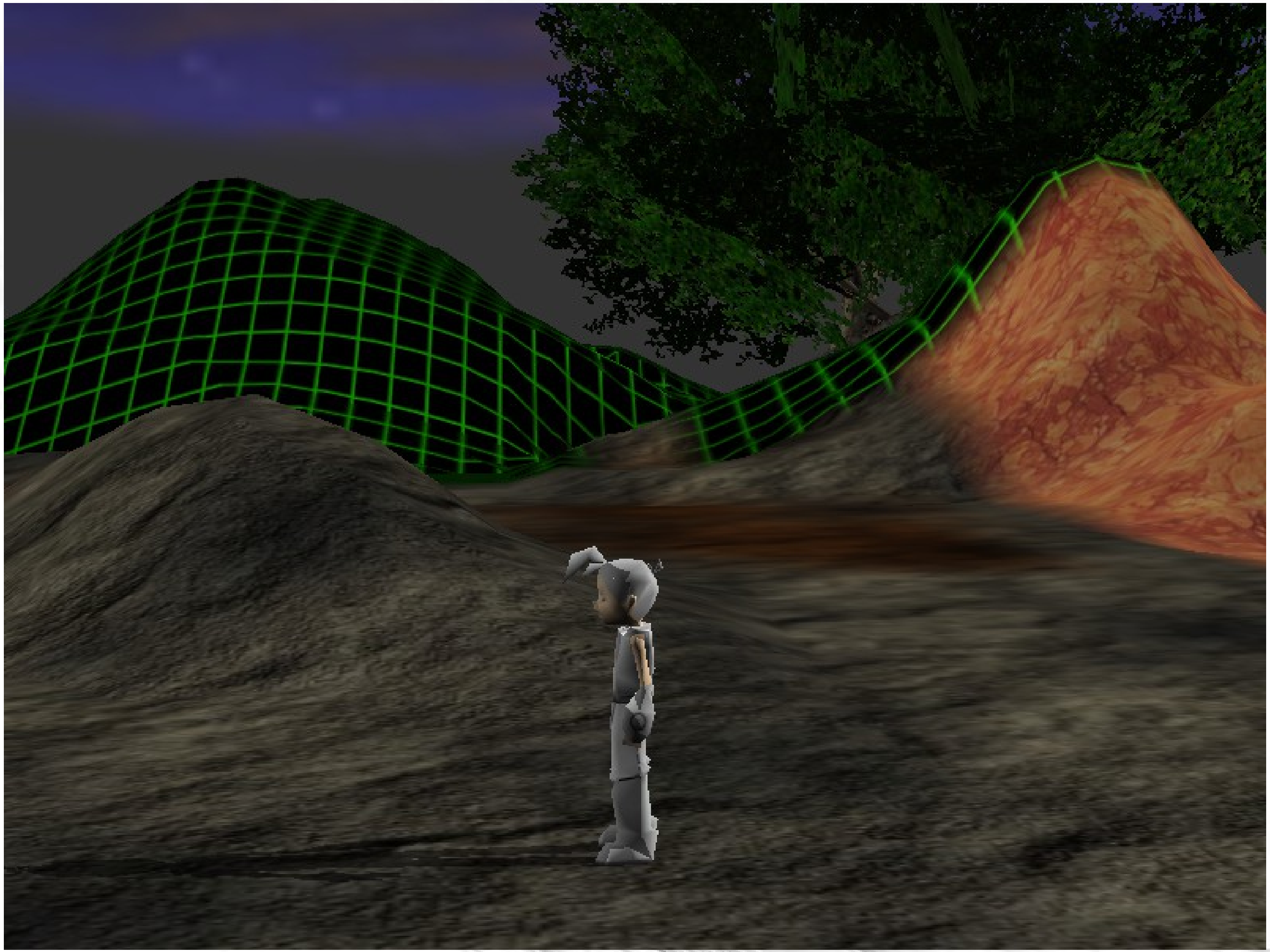
Speed Trails



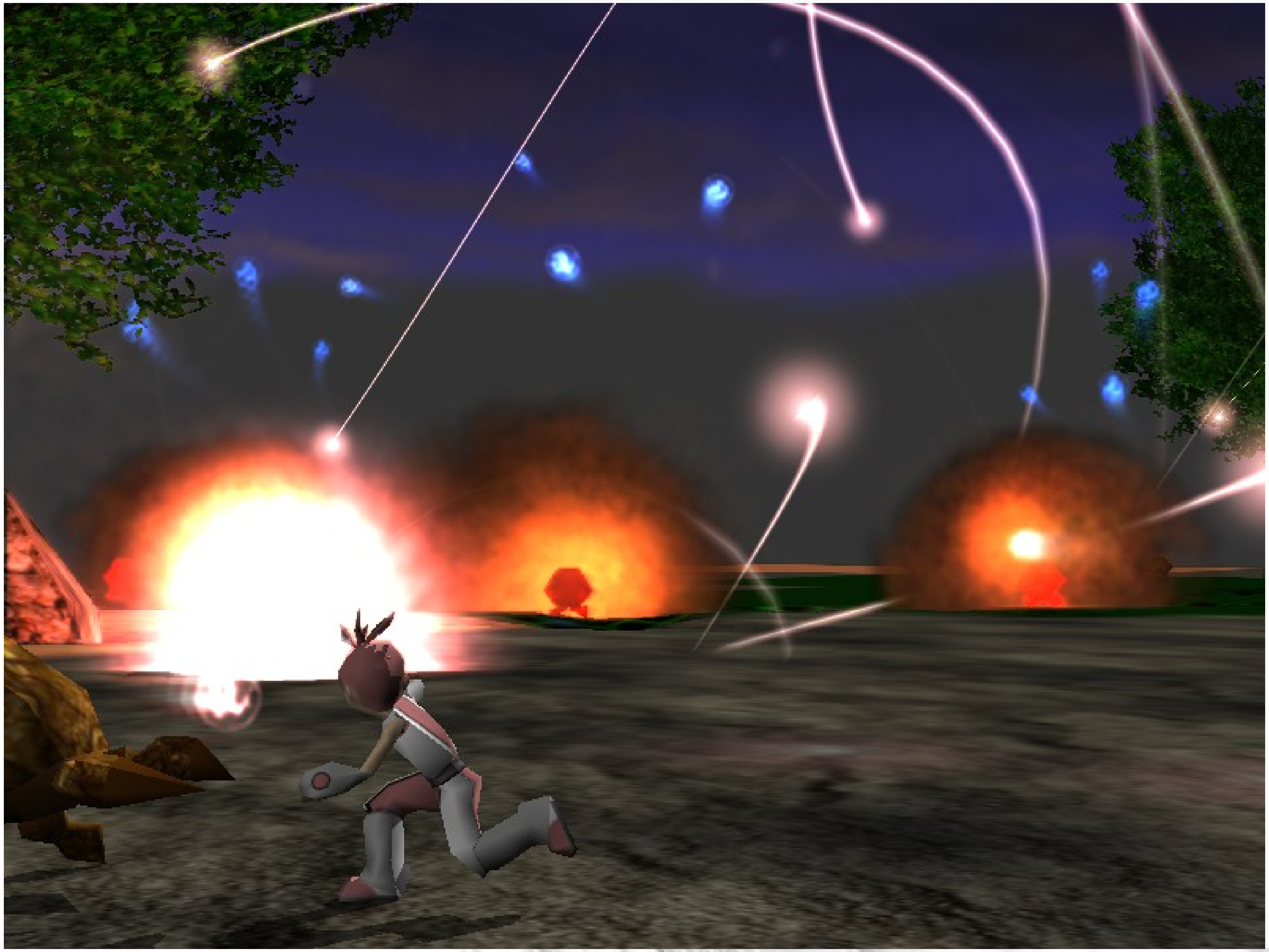
Toon Shading



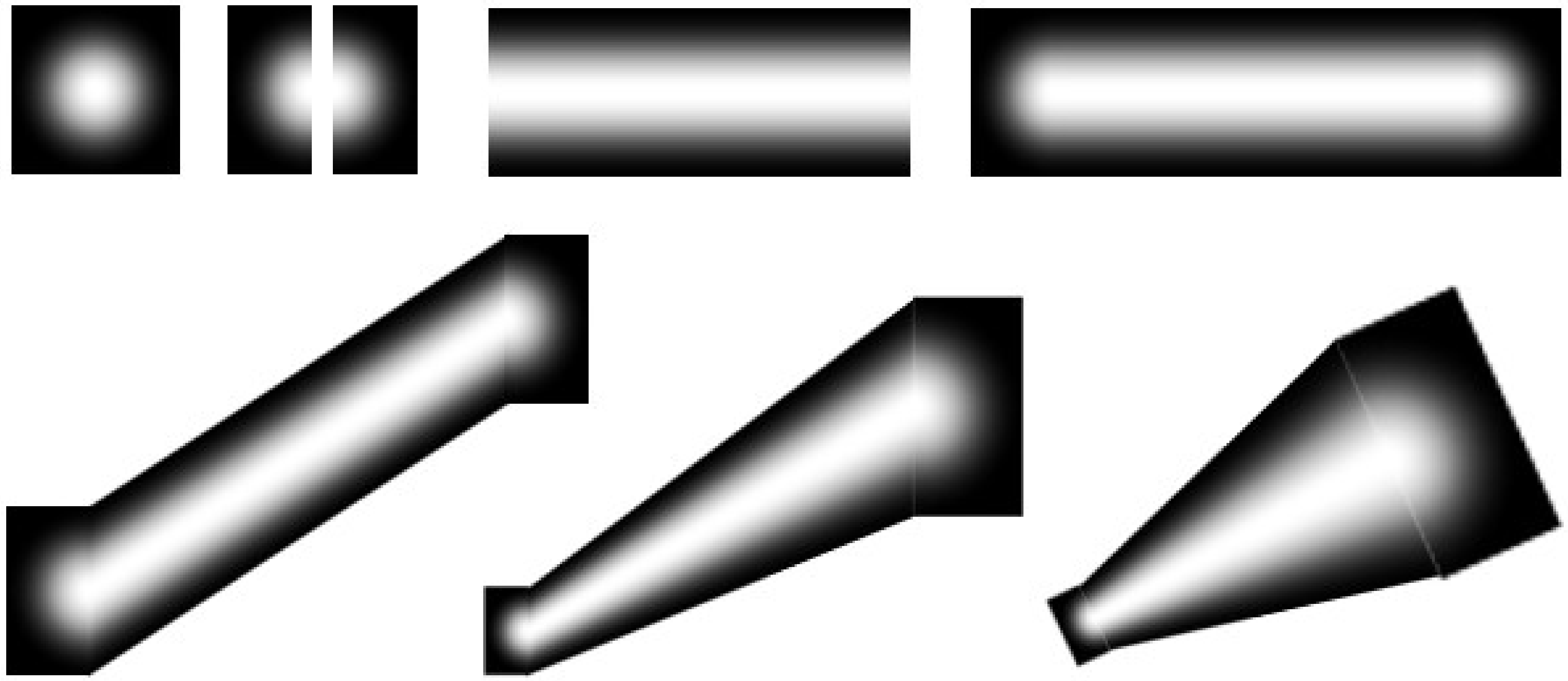
Terrain Deformation and Effects



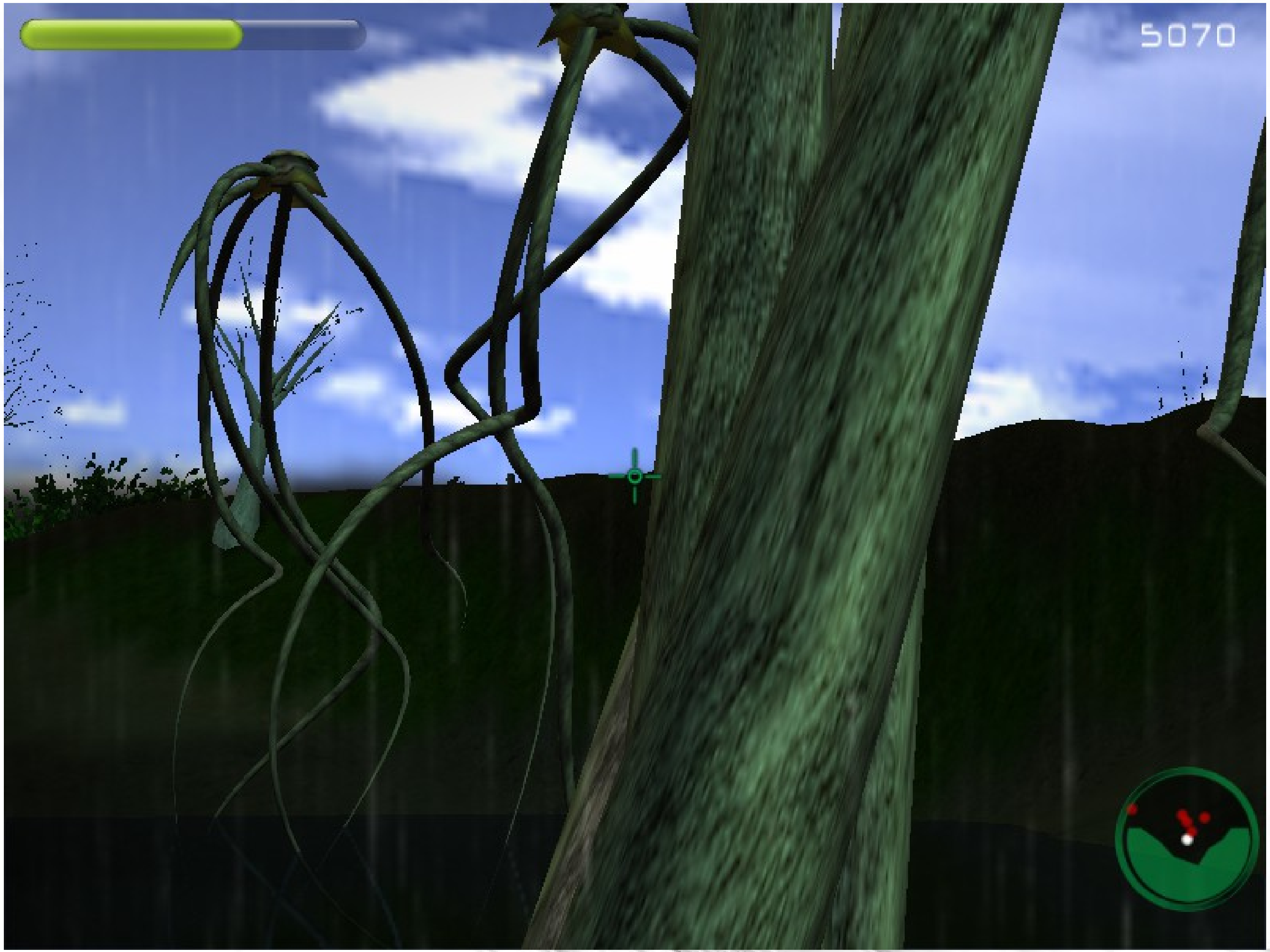
Particle Effects



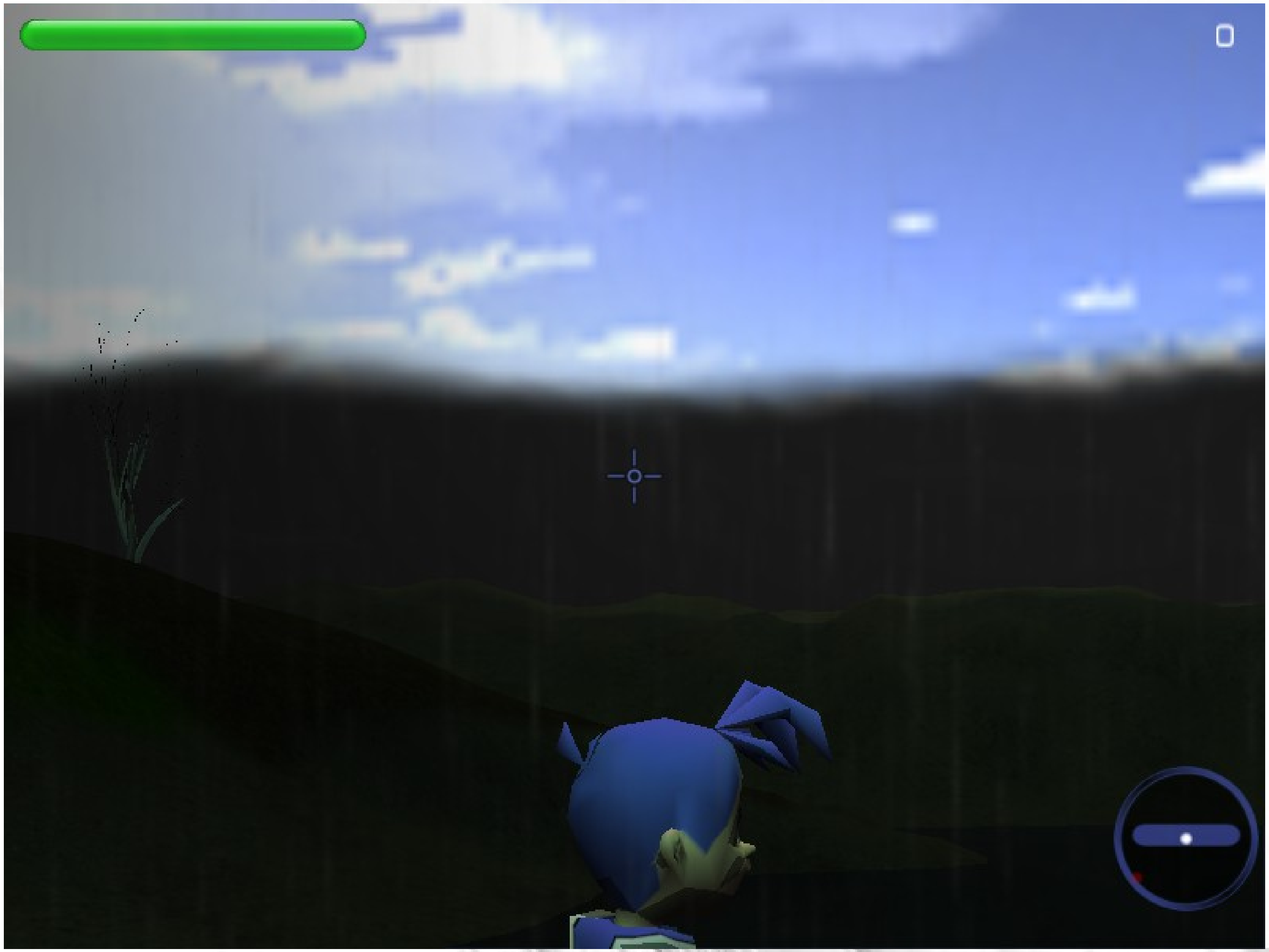
Laser Bolts



Vines

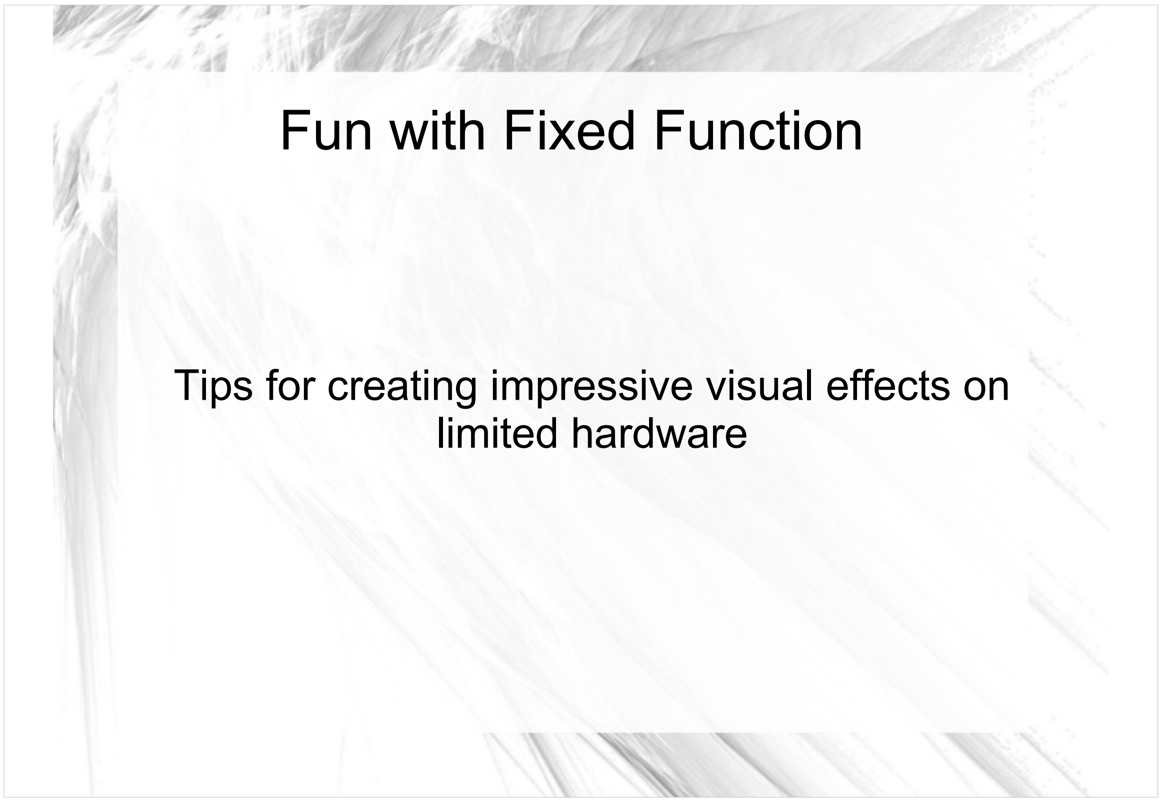


Full Screen Effects



Conclusion

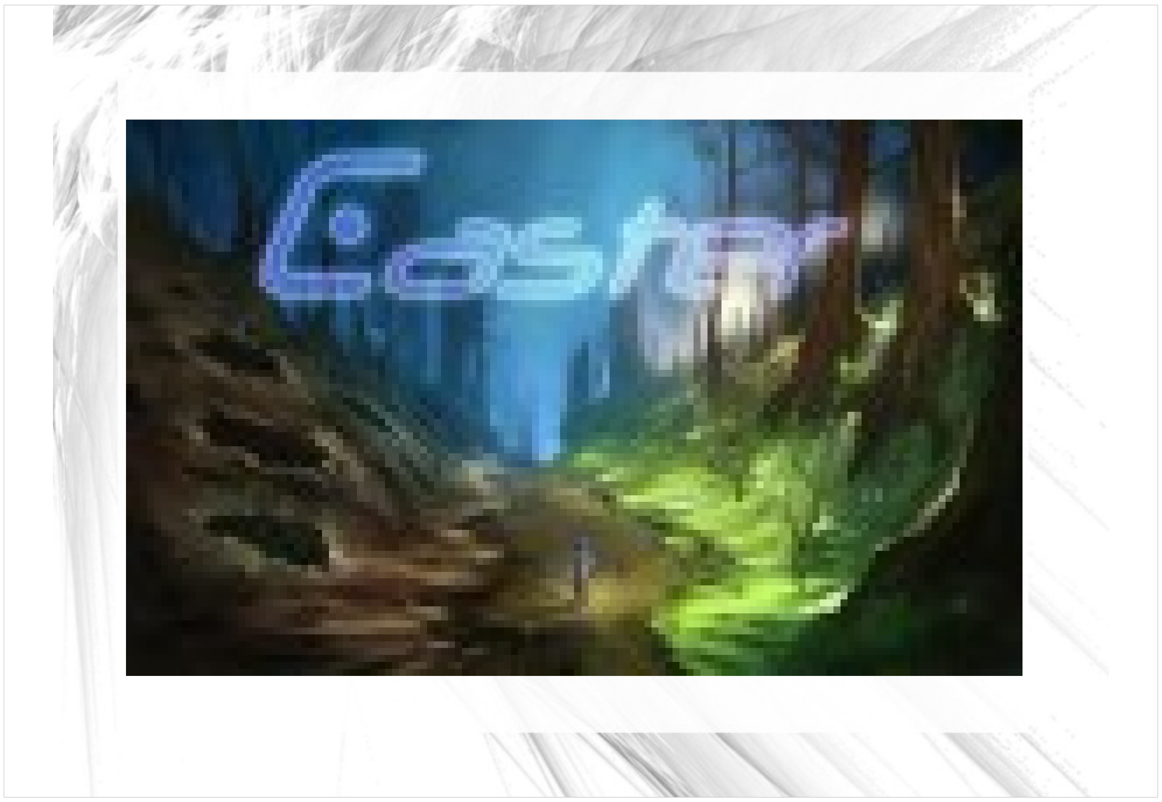
- Consider the end goal and don't get blinded by paths to get there.
- Faith Vision and Cleverness go a long way.
- You can probably figure a way to pull it off... or something that's really close.



Fun with Fixed Function

Tips for creating impressive visual effects on limited hardware

This talk may be a bit dated because of advances in hardware putting shaders into cell phones, but hopefully you will be able to garnish some ideas from it.



In 2004 started on the full 3d version of the game. Shipped it in 2009. It has since been ported to Mac, Linux, iOS, WebOS, halfway to DreamCast, and I'm finishing up an Android port.

My university education had a heavy emphasized in computer graphics and vision and I wanted to have some really cool visual effects in my game.

I also wanted to get my game in as many hands as possible (often called reach)

The computer I had to develop on was a Dell Inspiron with a GeForce II Go with 16 MB of video memory so I was bound to fixed function.

The game community at that time wasn't bothering with many special effects for fixed function because they were all excited about shaders. Seeing cool visual effects with shaders gave me faith that I could pull off similar effects in fixed function.

I was also developing principally on Linux and liked the idea of being cross platform. So Caster was written fully compatible with OpenGL 1.1 and due to some performance issues, doesn't make use of the multitexturing extension.

Inspiration and Faith

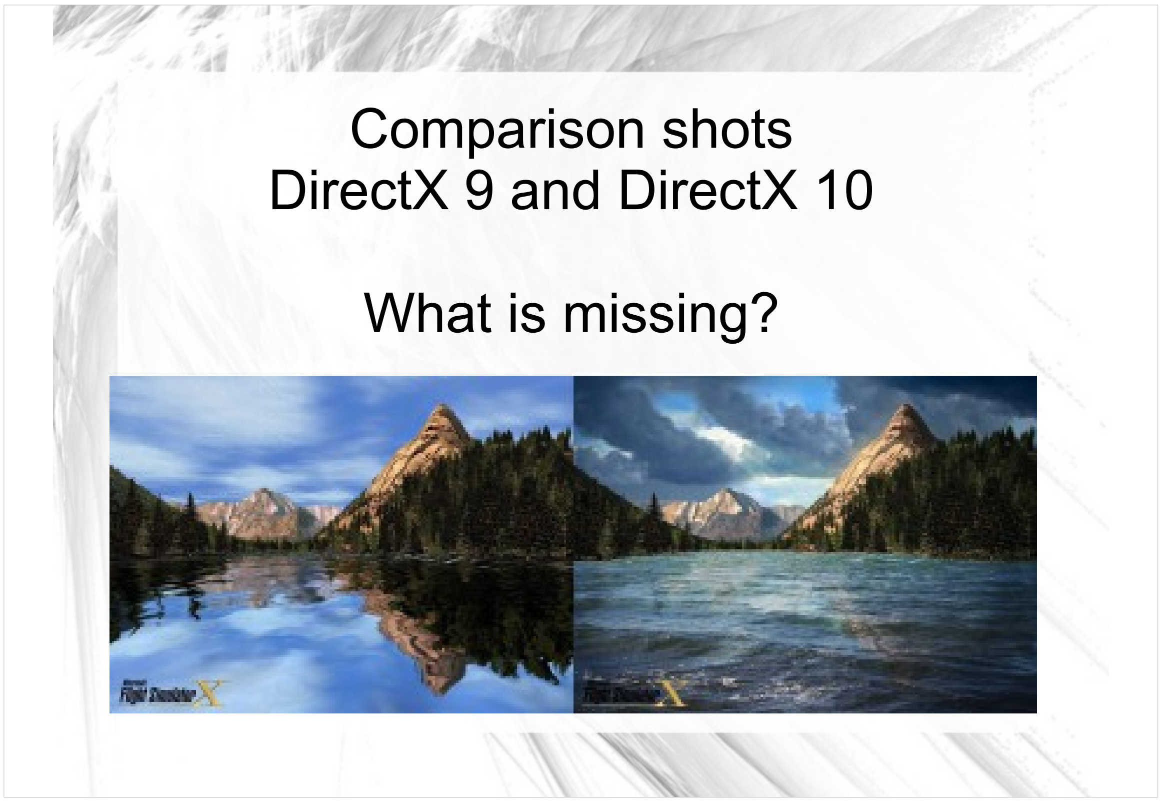
A coworker of mine, Peter Freese, had an experience where his studio had seen impressive screen shots of a rival studios video game. It contained all sorts of neat visual effects to the point where they were stumped as to how the studio pulled them off.

Peter and his group set to work trying to figure out how they did it and adding those effects into their own game. It was difficult, but they finally prevailed and were able to have similar visual effects in their own game.

Only later did they find out that all the interesting visual effects from their competitors game had been painted in on top of a screen shot.

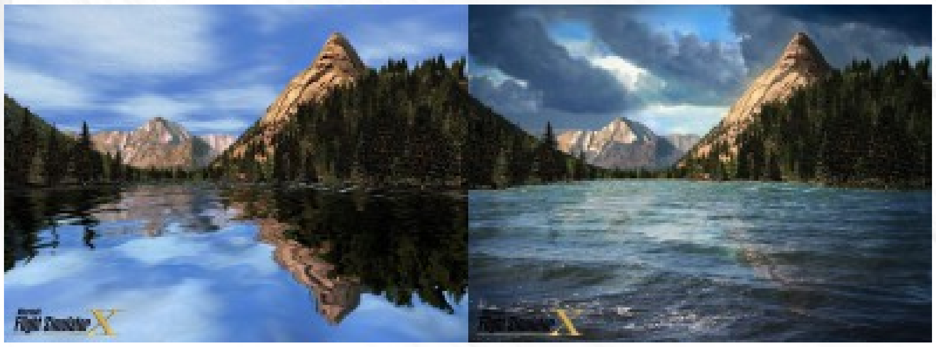
So here is an eternal principle that you can take away from this talk: Faith is a powerful thing. It's the enabler to get things done. You only ever do things because you believe they can be done. If you don't believe it can be done, you can't do it. That's generally why those with positive can-do attitudes often overcome tremendous odds to do amazing things.

On Caster, I got a lot of my inspiration from the European demo scene. You don't know how they do cool effect X, but you're convinced that it can be done... even thought the demos often lie cheat and steal to pull them off... Which is totally encouraged in games!

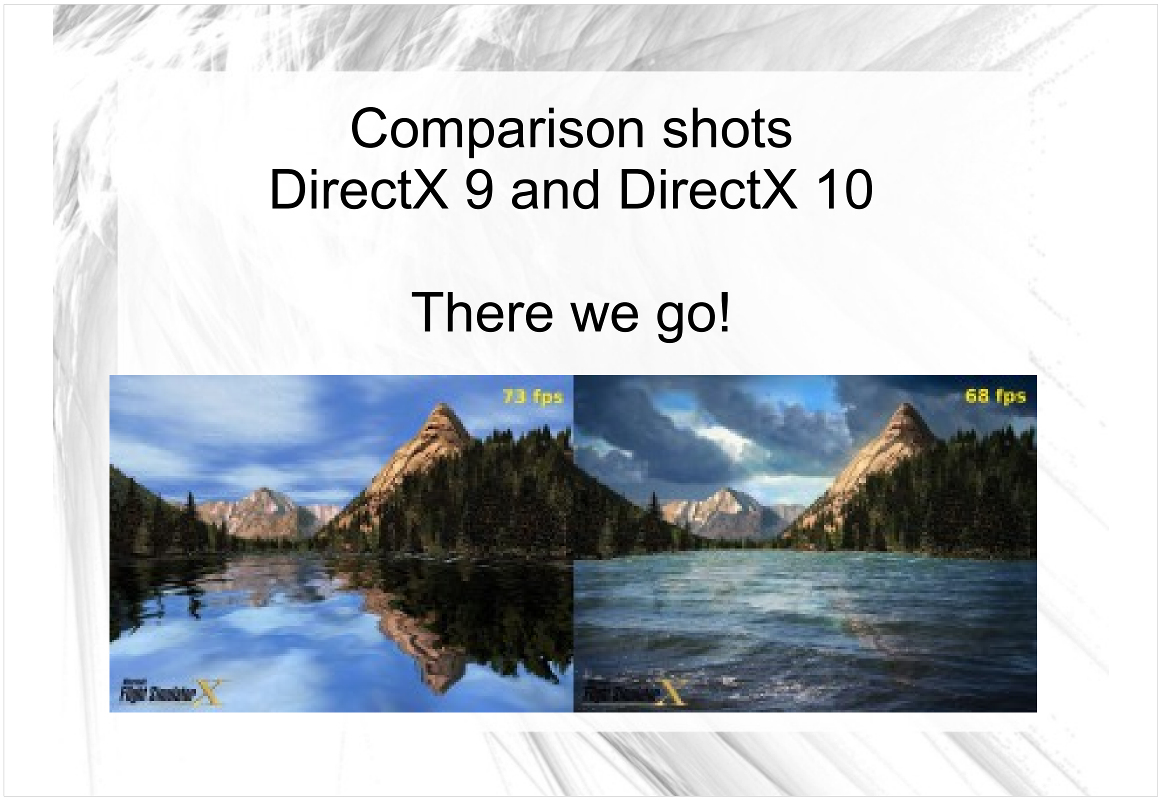


Comparison shots
DirectX 9 and DirectX 10

What is missing?

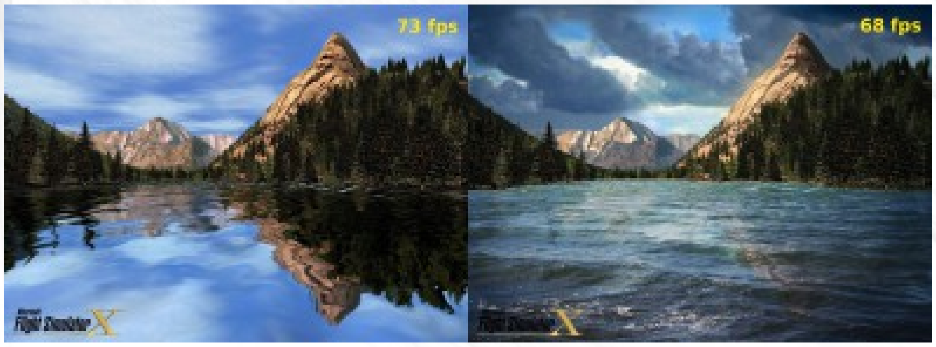


- Sometimes people get caught up in technology and miss the goal of what they're trying to accomplish.
- I once had a boss say "lets add some PS 3.0 shader effects." Wrong request. "I want this visual effect." is the correct request.
- I’ve often seen even seasoned veterans in the industry get hung up on and start preaching hardware specifics rather than focusing on what they want visually.
- New and better graphics hardware will NOT make things look better. It does NOT let you do special effects you could not do before.
- Image to image comparisons between new versions of direct X or the latest graphics cards are meaningless... without also showing the frame rate or talking about how easy it was to pull off.

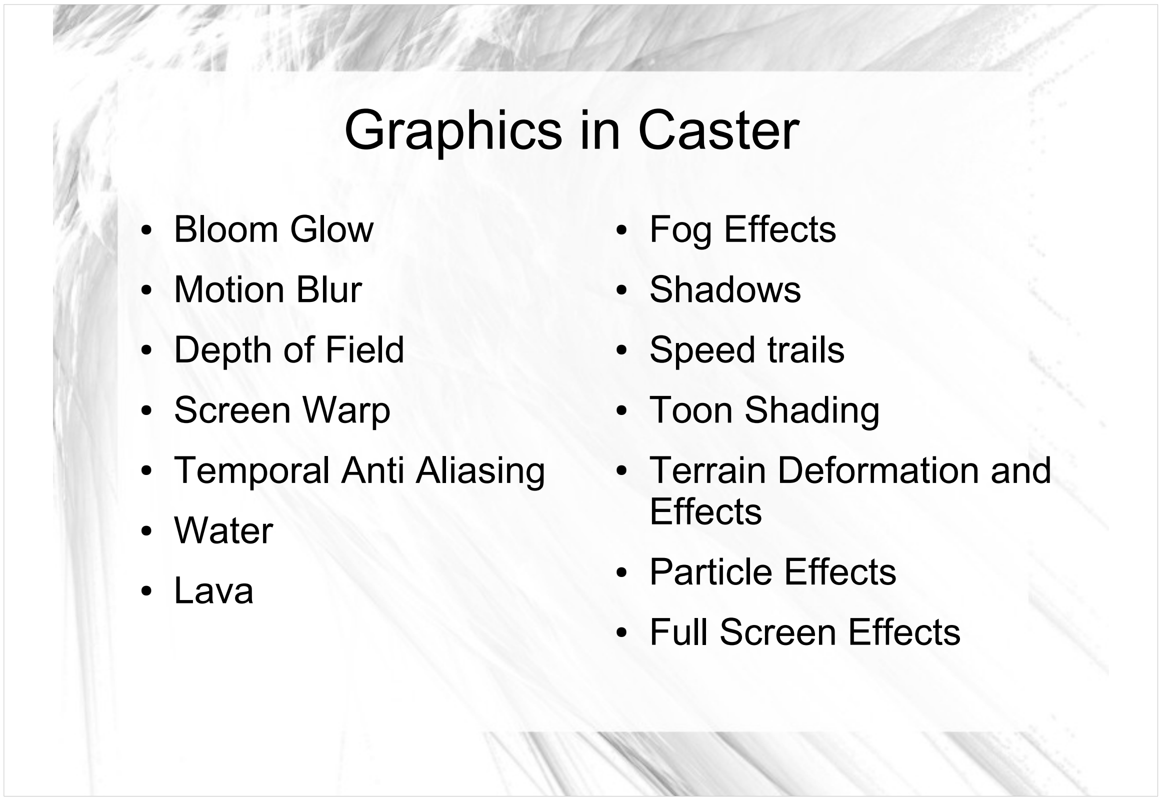


Comparison shots
DirectX 9 and DirectX 10

There we go!



- And that's all graphics hardware and new versions of your graphics API are. POTENTIALS FOR PERFORMANCE INCREASE AND EASE OF USE. NOTE: Do not discount the EASE OF USE bit. This has been one of the most significant things about new graphics and CPU hardware. You don't need to
- So if you find that you are rendering super sweet effect X but the perf is in the tank, You might want to reconsider other approaches.
- Often, cleaver tricks will allow you to do super sweet effect X or something very similar with much better perf and better reach (IE lower end graphics hardware).
- Graphics effects is all smoke and mirrors so cleaver tricks that make things look better are encouraged.
- I feel like when a new technique or hardware API comes out, there is a lot excitement in the community and a lot of pressure to jump on board and start using it without objectively weighing in what it gives you and what the cost is:
  o in performance and or reach which are a big BIG deal. Perf for game experience, Reach for sales.
  O Easy question to ask about performance, not so easy to evaluate in the PC world of numerous hardware configurations.
  o Generally, make the game look the best you can on min spec and add a few extra bells and whistles for higher end machines. Be careful about adopting an approaches that automatically cuts you out of a large share of the market.
    - For example, writing your engine to require DirectX 13 when by ship time there is only a 1% market share of that installed.
- On newer graphics cards, I have often seen older ways of doing things outperform the new special extension with barely noticeable differences in quality if any.
- Also there is a tradeoff of the actual impact of the effect and the cost. Maybe it's just not worth it or maybe only do it every so often in balance with something else.
- DON'T FORGET CLEVER ART! It goes hand in hand with clever graphics programing tricks and the distinction between the two is blurry at best.

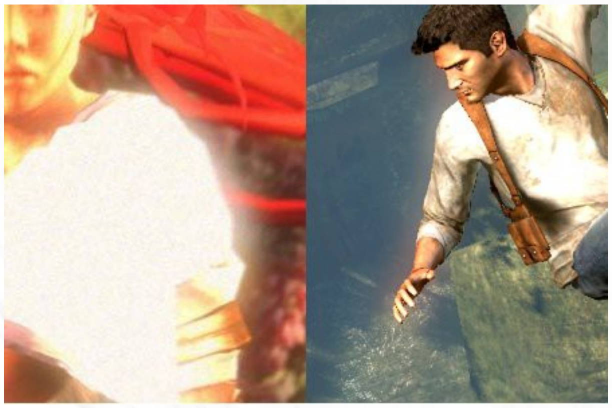


Graphics in Caster

- Bloom Glow
- Motion Blur
- Depth of Field
- Screen Warp
- Temporal Anti Aliasing
- Water
- Lava
- Fog Effects
- Shadows
- Speed trails
- Toon Shading
- Terrain Deformation and Effects
- Particle Effects
- Full Screen Effects

Techniques I'll be covering

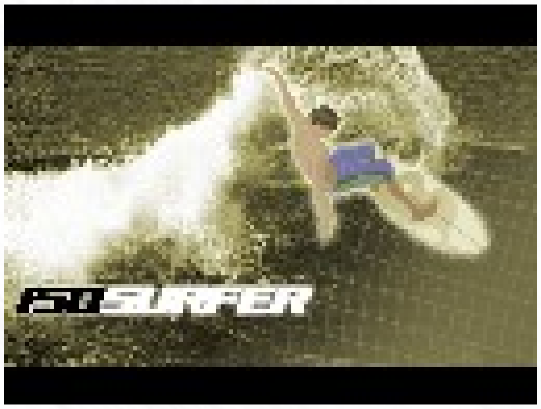
Bloom and Glow



Do not want

We wanted to do some nice bloom / glow effects for illuminated items in the scene.
A lot of people were doing bloom by bluring the brightest parts of the image. We didn't want white shirts to glow.
We thought of trying to filter off the brightest bits of what was in the buffer using some copy function or other that I can't remember. It wasn't working well.
We thought of rendering the whole scene darker so that things half bright would be full bright and full bright would be candidates for bloom.

Bloom and Glow



[isosurfer]

Then it dawned on me. Why not render everything with black materials and keep the specular rendering normal since specular is emissive light bounced back into your eye.

Then we could render the resulting texture multiple times shifted over the screen to create a blur.

We did radial blur by scaling the texture from the center of the screen.

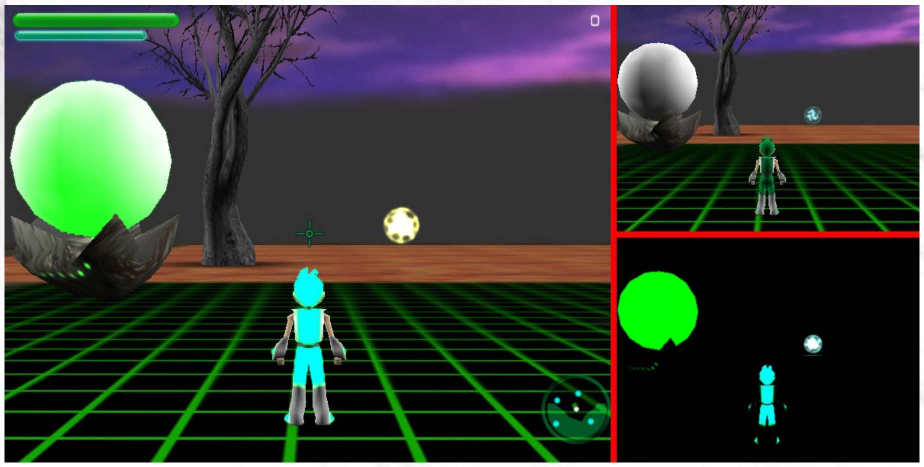
I incorporated the idea back into Caster and realized I could use this to render anything that I wanted to be emissive / glow. Explosions, terrain impacts, the main characters suit, enemies when hit, lava, acid, beacons, resurrected trees, energy pickups etc.

I could also create a “glow mask” for characters which was a texture that was completely black except for the glowing bits with their color.

I also realized that I could render certain things only in the glow pass to give them a smokey feel. (mist, acid, beacons)

Glow pass of solid red that faded off to black over time to show when you hit enemies.

Bloom and Glow



Inside of Caster, multiple passes of the glow texture to bloom it out was killing fill rate and overall performance, so I had the idea to spread the cost over multiple frames. This resulted in a misty glowy trail effect in the game that ended up being a nice feature. Spreading the work over frames is still a common approach in modern hardware, for example in determining scene intensity for iris adjustment where you don't need an immediate result.

Render the world with glow.

After rendering the world with glow, draw the last frames glow texture on top of it but set the transparency to something like 70 percent.

Copy the results back into the glow texture.

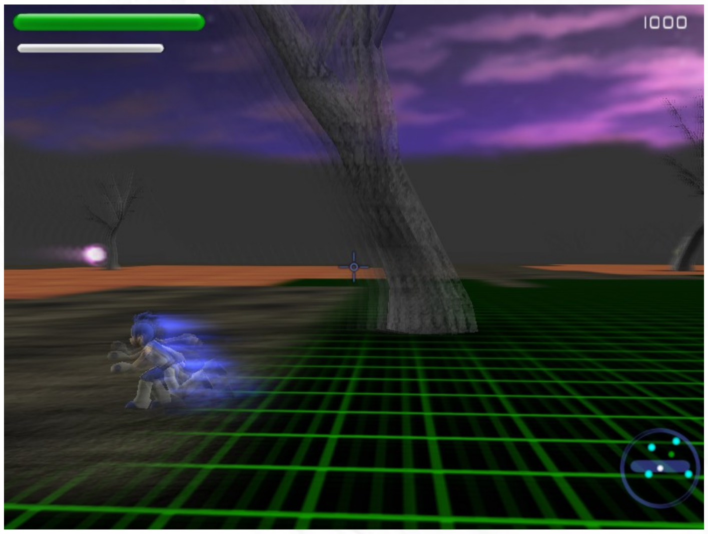
Render the world normally

Now set your blend mode to additive and render the glow texture over the whole screen.

When we went to first gen iPhone / ipod touch, it was too expensive to copy from the back buffer and do extra full scene passes, but glow had become an important game play element. It was used for example when you charge up before you attack and when you hit enemies.

To keep those features for selected objects and effects, I did a second pass using additive blending on the glowing geometry with an additive only doing a z test equals check in the depth buffer.

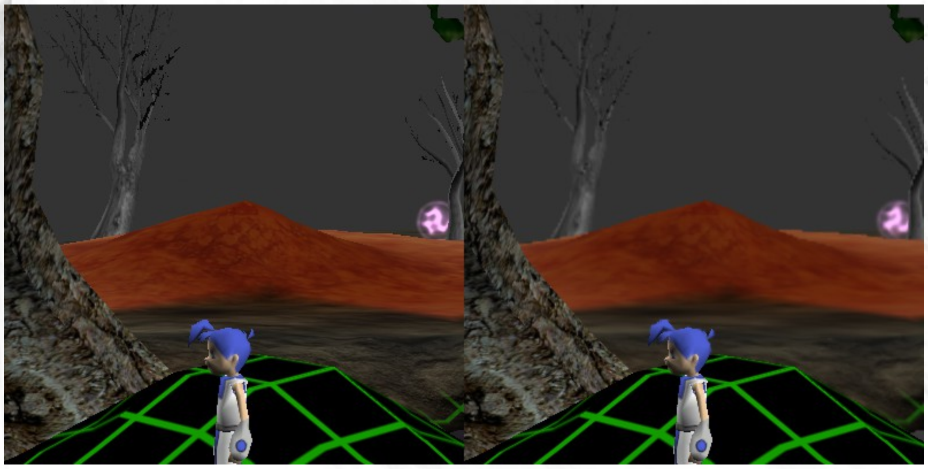
Motion Blur



On top of this render the motion blur capture you took from last frame but set the alpha to something like 70 percent or whatever you want. The less you blend in, the less blur residual you will have.

Capture the world back into the motion blur texture.

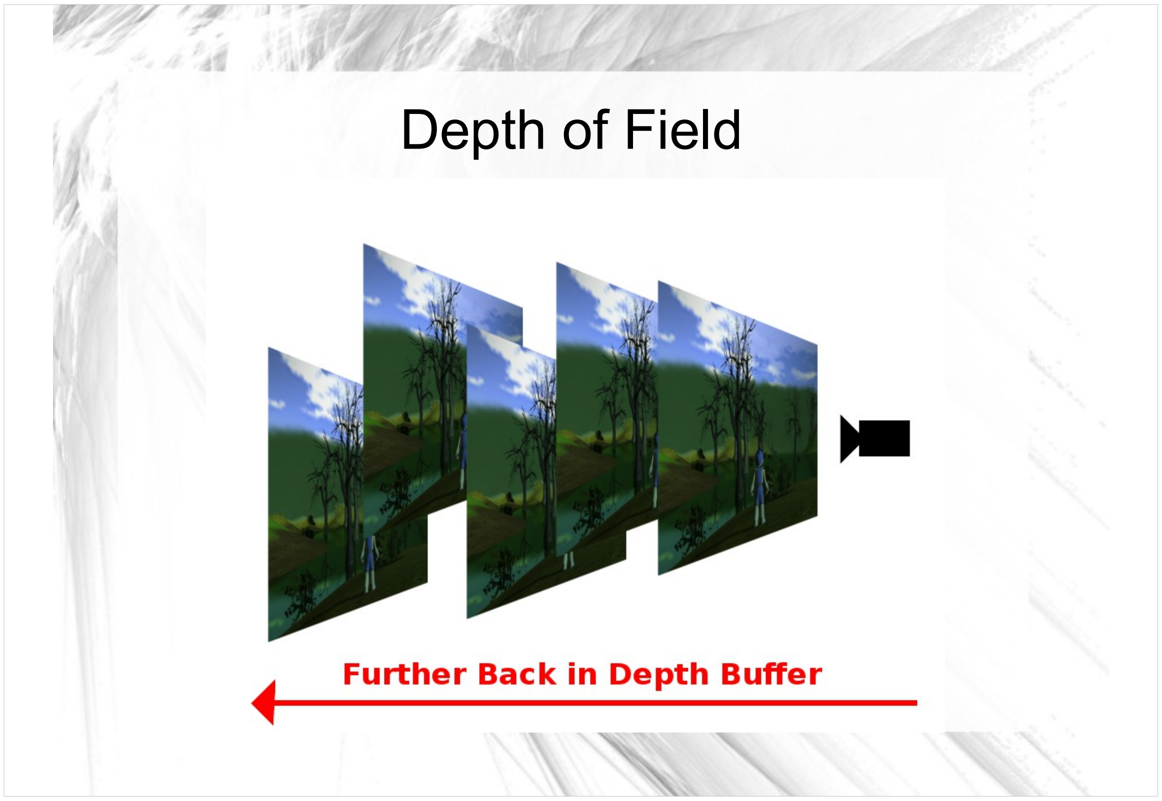
Depth of Field



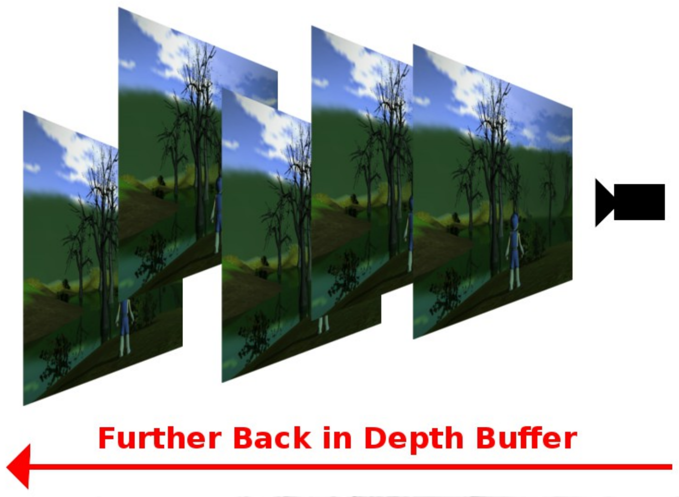
We wanted to do depth of field in Tiger Woods golf but we didn’t have read access to the depth buffer.

A common way to do depth of field is to blur pixels further away from the focal point. The further away the pixel is, the more blur it gets.

After pondering the problem, I came up with a technique to address this. I knew I could do full screen blur by simply rendering the screen multiple times on top of itself, but this would blur all the pixels on the screen equally.



Depth of Field



So I came up with the idea to selectively blur pixels based on depth by keeping the depth buffer from the main scene render and rendering a full screen quad set in the world moved back along the z axis. Each time I would do an offset render pass, I would shift the quad further back away from the camera. This way, pixels the furthest from the camera would accrue the most passes while those closer would have less.

Set the depth test value to be where the focus is. With the depth buffer enabled, render the full screen shifted slightly while moving the image further back from point of focus.

As you can imagine this technique was fill rate intensive which made it impractical for lower end machines when pushed to an extreme blur sample size of 25 or so.

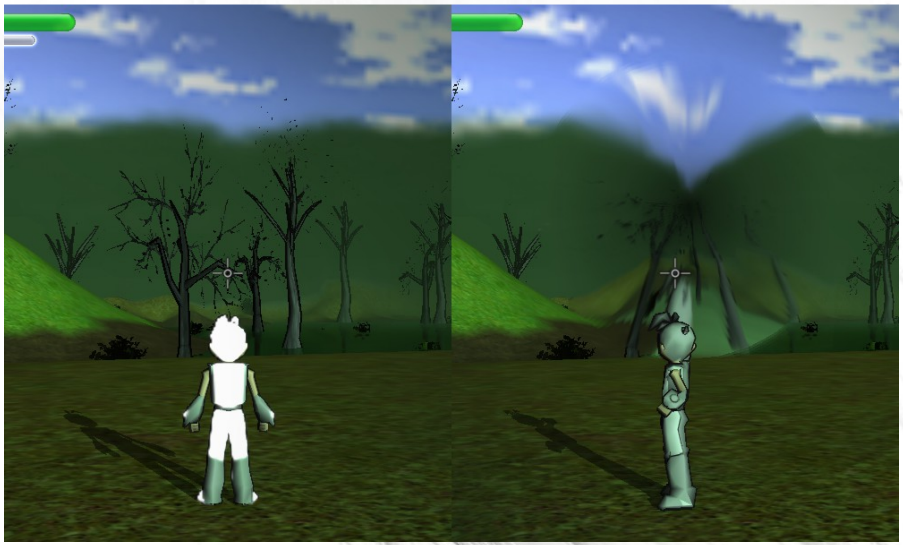
We ended up using it for cut scenes in Tiger Woods Golf.

In Caster, I used the full screen buffer also used for the warp effect.

Drew a single quad aligned to the camera viewport a certain away from the camera in the direction the camera was looking.

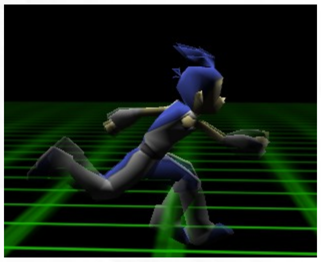
You can use alpha to transition to more or less DOF.

Screen Warp



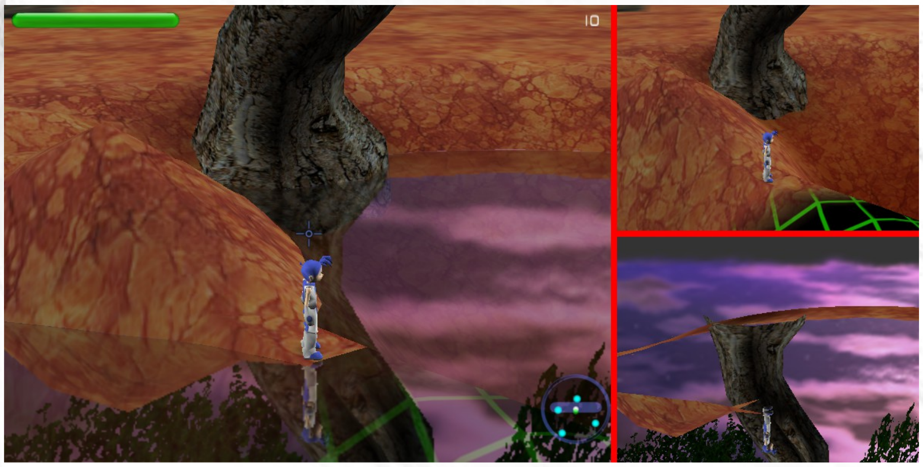
Render to back buffer and copy to full screen source.
Texture coordinates in screen space. Stretch to middle. Tint middle vertex black.

Temporal Anti Aliasing



This one requires two buffers, or you can just think of it as motion blur above but shifting your viewport by half a pixel every other frame.
Render the world
Copy this into a capture buffer.
On top of what you just rendered set the alpha to 50 percent and draw the buffer you captured last frame.
More details here: http://elecorn.com/blog/2008/10/super-fast-anti-aliasing-for-free-well-almost/

Water



[demo_water_fog]

Stencil reflections

Render the world flipped around the water plan. Optmizations include only rendering the sky box or just the terrain.

Copy the results to a texuture.

Render the world normally and keep your depth buffer.

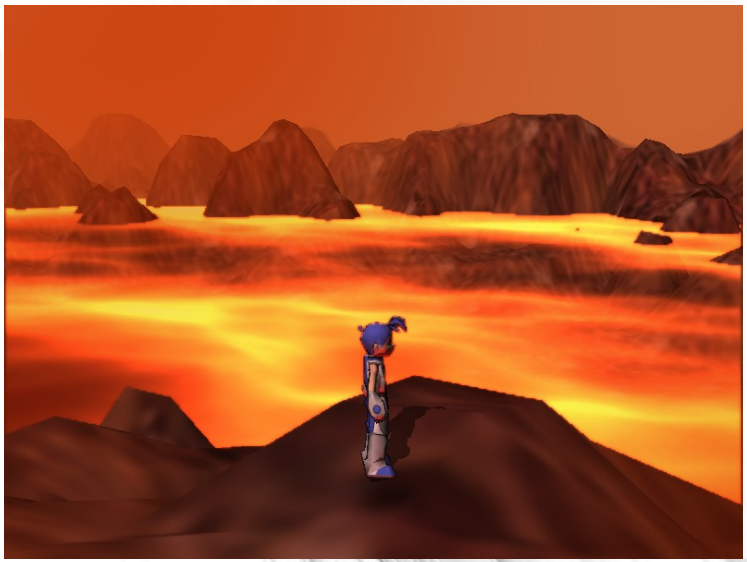
Render a quad for the water plan into the stencil buffer.

Render flipped world texture over the whole screen.

The alpha for the color is determined by your camera’s view angle with the water to make the water more opaque or transparent (FRESNEL effect). Steeper angle = more transparent.

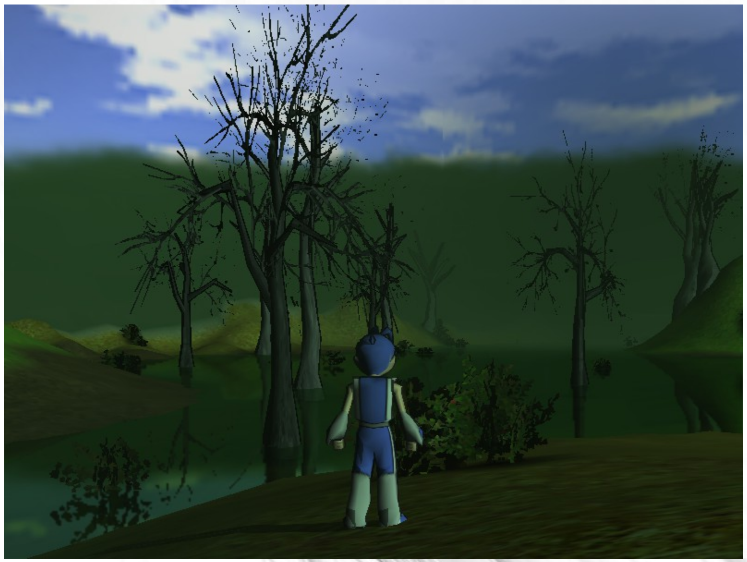
Underwater (full screen tint) and motion blur. Aggressive fog. Height based color tinting of the terrain vertices.

Lava



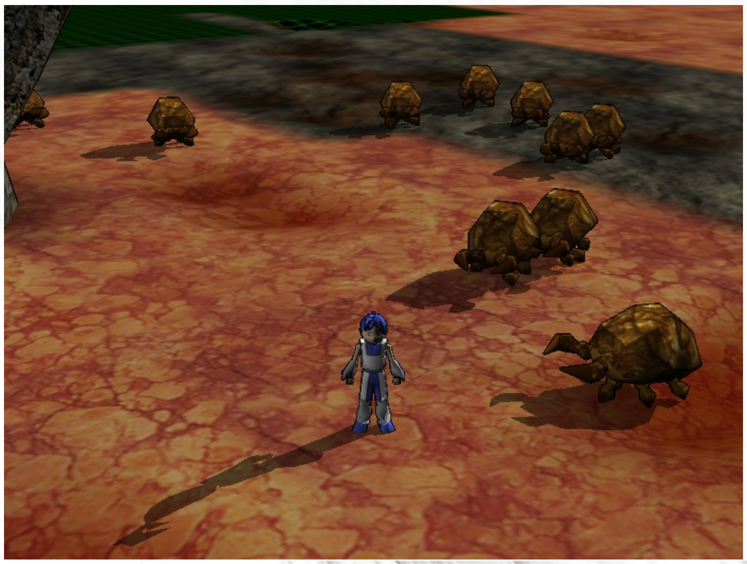
[demo_lava]
Decided to make only one "water type" per map. WATER, ACID (LAVA).
Lava via glow only pass, acid effect.
Mist via same idea.

Fog Effects

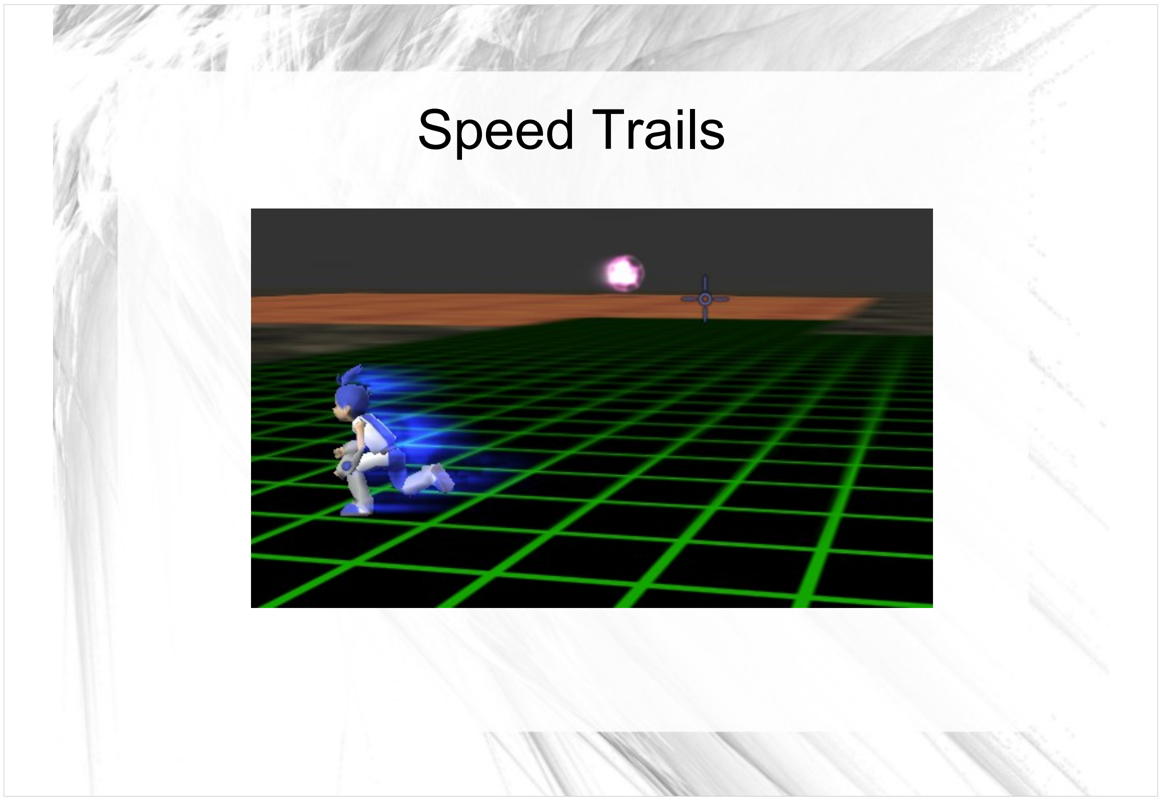


[demo_water_fog]
Terrain color blending at the edges.
Tinting via mask on the sky box.
Fog on water using a white circle texture with the middle cut out of it. The base color for the verts was the fog color. I would scale the texture to the size of the fog fall off point. As you jumped higher, I would scale it down. Border repeat in the texture settings.
Vertex based distance fog.

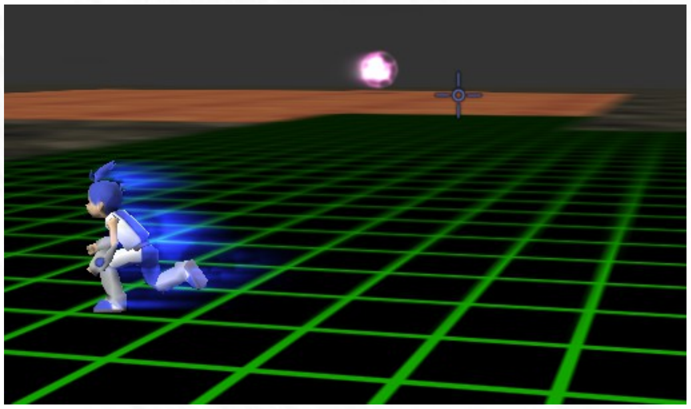
Shadows



Create an edge list for each object animated or not.
Every update to animation (usually once per frame, NOT once per render pass), collect the edges that represent the silhouette of the model.
Render from faces of the extruded volume into the 8 bit stencil buffer incrementing the value in the stencil buffer.
Then render the back faces into the stencil buffer decrementing the value.
Then do full screen alpha quad using a stencil check for values greater than zero and render shadowed pixels.
Upsides: Self shadowing clean sharp shadows.
Downsides: Same with all shadowing techniques, if terrain is not shadowing then you see artifacts. Stick with blob shadows by default, keep main light angle fairly vertical.
Circle Shadows:
Grab terrain vertices around shadow blob. Draw those verts with new texture coordinates that map to the shadow texture.



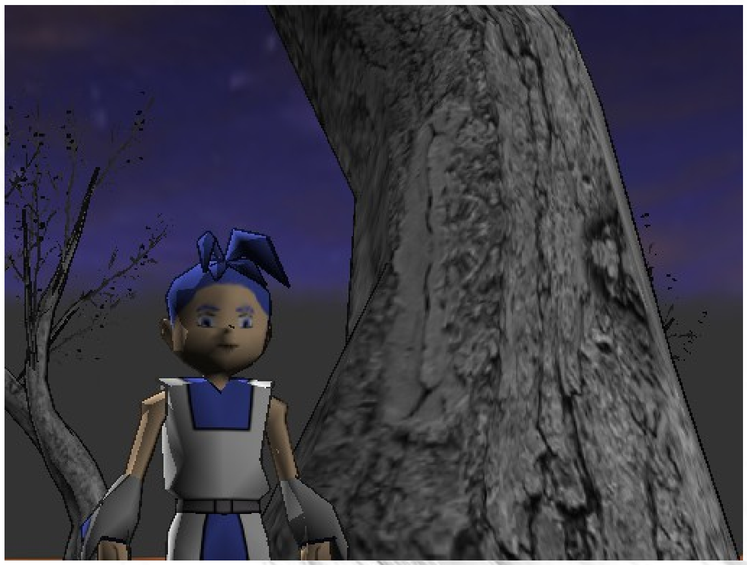
Speed Trails



Used stencil shadow silhouette to do speed lines. Changed light source to be in front of player's run direction.
Rendered quads at solid color on the silhouette and alpha 0 at the end.
To increase the illusion of speed:

- FOV change
- Full screen blur.

Toon Shading

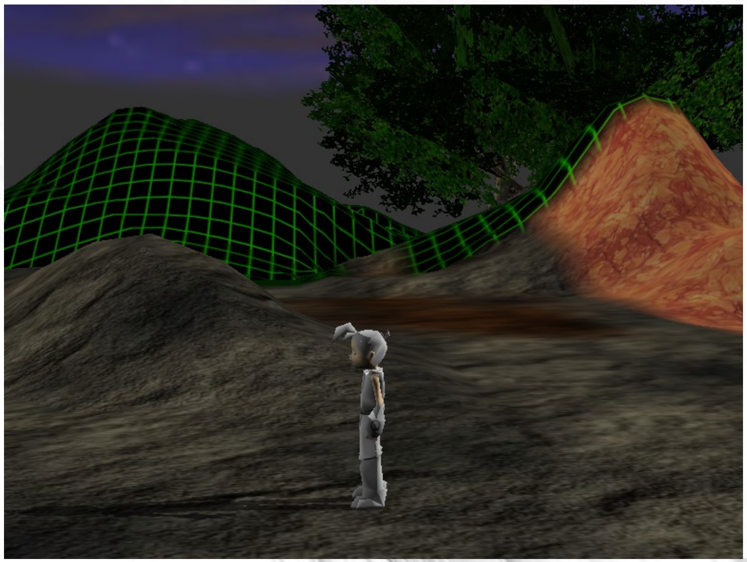


Thought it would be cool to do toon shading (it was the cool thing at the time). Other fixed function approaches rendered a scaled up version of the model with the culling flipped and colored black [PICTURE QUAKE].

Maybe could have used the silhouette idea as used in dash effect, but there was a cheaper way that put all the work on the GPU.

First, flip the culling of the models that have toon shading and draw them using GL_LINES Tinted / textured. Then flip the culling back and draw them normally [EXAMPLE]. You can use line hints ,but then you have an alpha sorting issue which may be hard to deal with in your opaque rendering pass.

Terrain Deformation and Effects



[demo_generic]

Terrain deformation

- Terrain is just a height map.
- Mountains vs. craters, invert terrain.
- Relighting the modified vertices after deformation.
- Glow on terrain when deforming.
- Camera shake.
- Vertices reevaluated every frame based on visibility.

Terrain Texture Blending

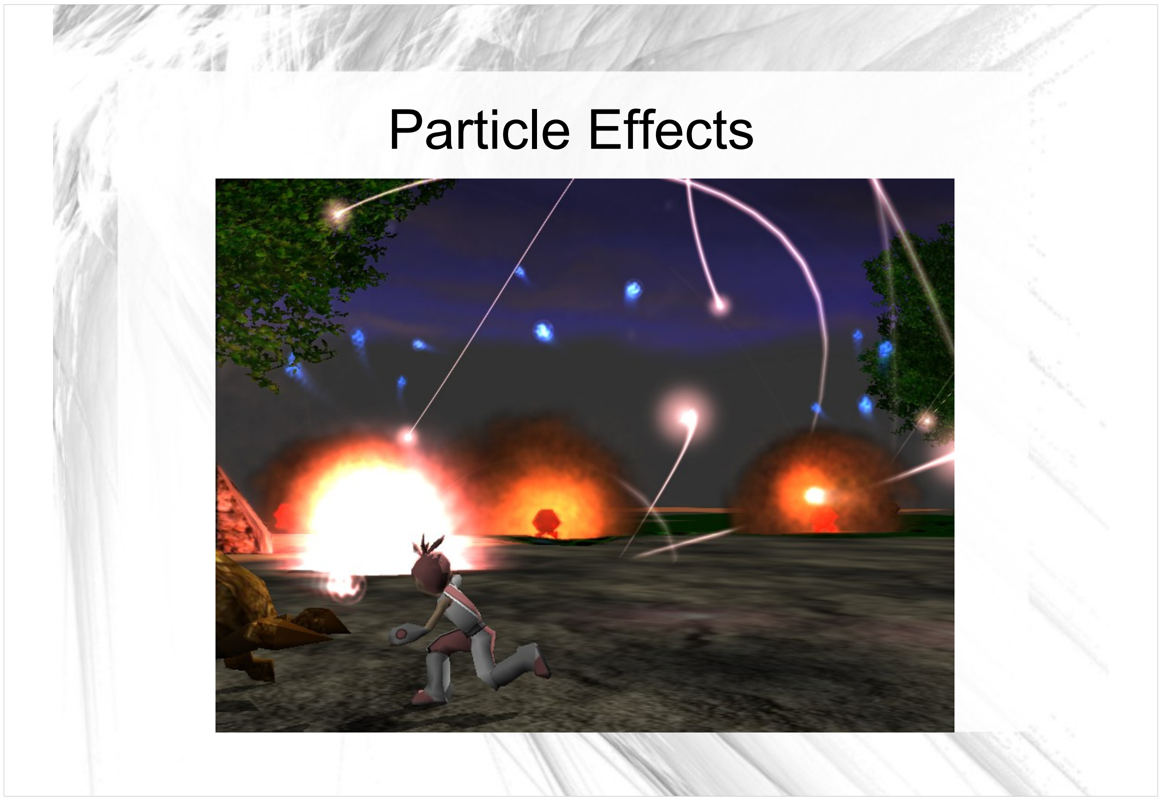
- Blending between different materials.
- we have a color map where each material corresponds to one color.
- We draw all the quads unblended and then draw alpha blended triangles in the transition areas.

Terrain lighting

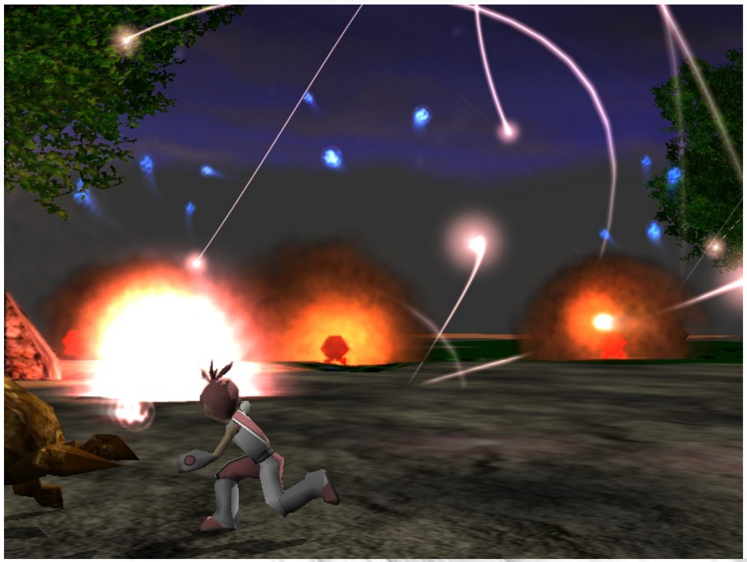
- Software lighting calculation. To mimic the hardware lighting math.

Terrain Ripple Effect

- Keep a bounds of the ripple area of effect, reset the verts after the effect has completed.



Particle Effects



[demo_particles]
No vertex buffers, no vertex shaders. All immediate mode.

Nothing fancy. Just manipulating each particle on the CPU. Created buffers on the CPU and sent them over to the GPU.

Seeker effect

Lightning effect

Additive blend / glow pass, smoke.

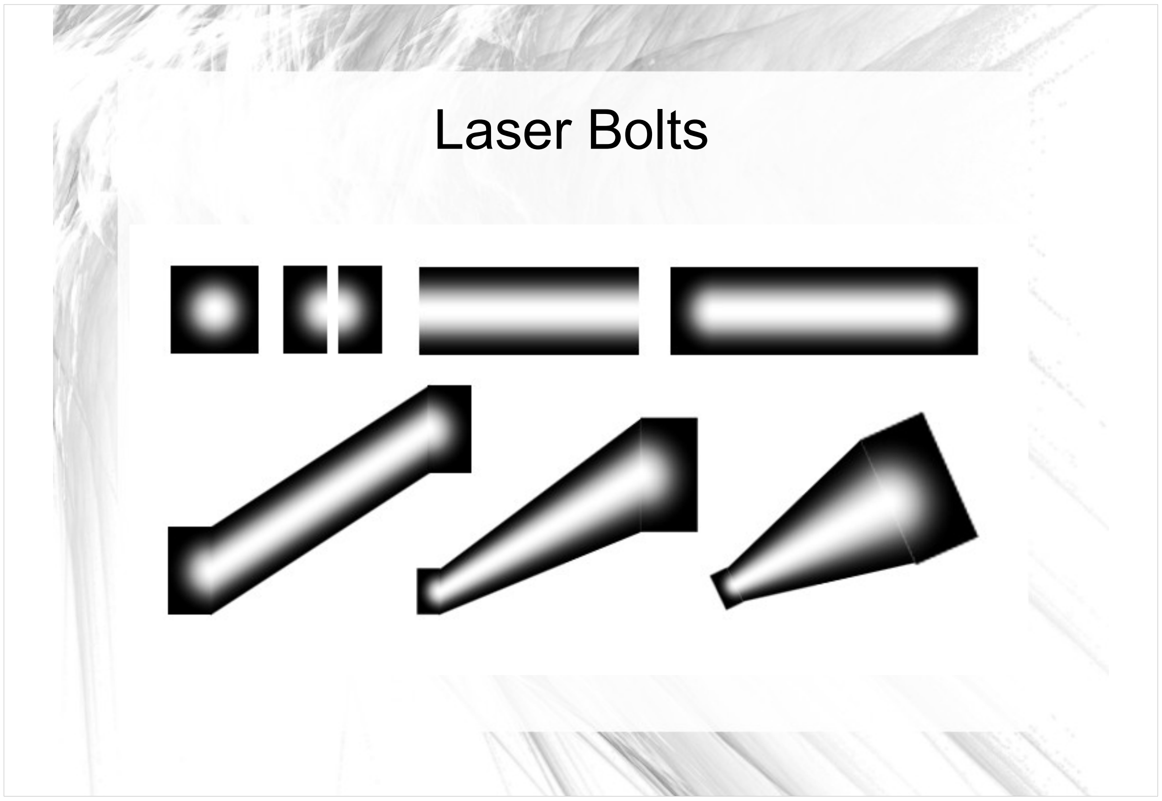
Multi-target and stun effects paths:

B-Spline curves evaluated at a time T along the path.

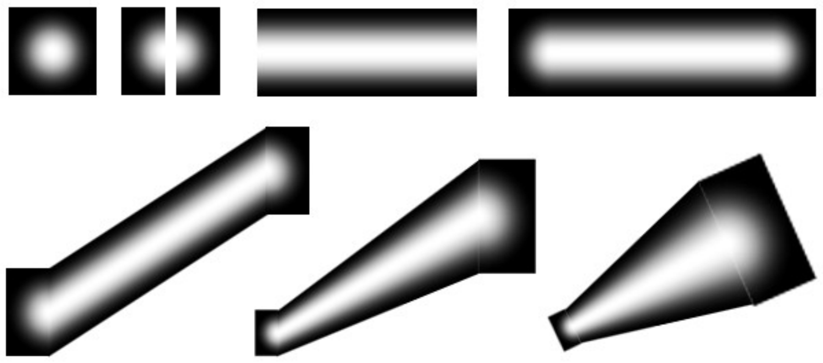
Rendered into glow pass.

Additive rendering.

Used sprite01 texture.



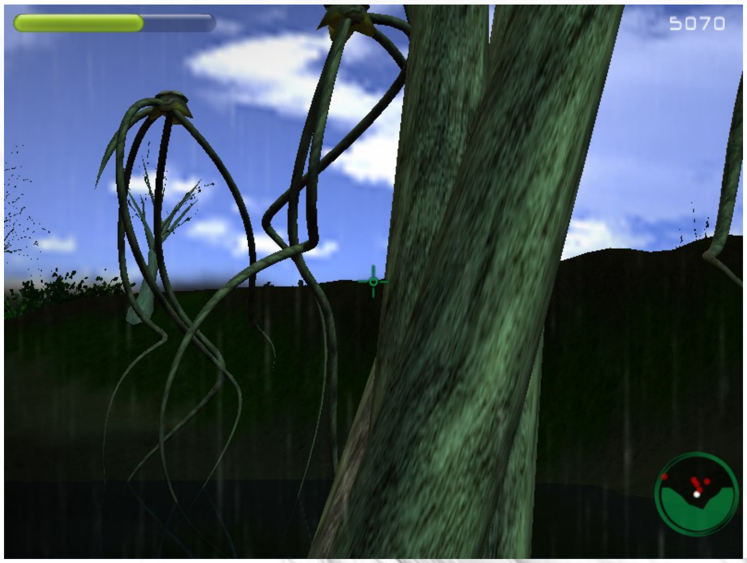
Laser Bolts



Laser Bolts:

Project endpoints to camera axis. Draw quads back in 3D.
Shadow decals (that glow).

Vines



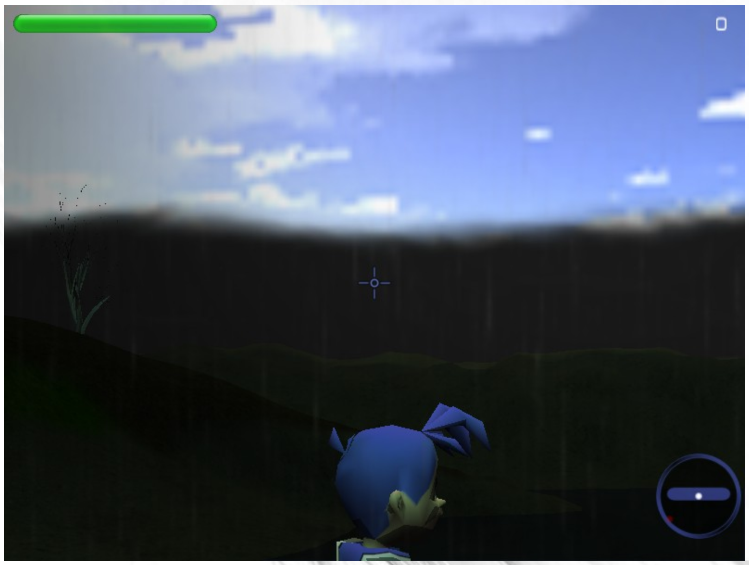
Vines:

Bspline paths like the other things. Evaluated along the path creating triangles around the path to make spheres.

Since the vine source and end points were changing, I did reevaluate the curve each frame.

For wrapping around the player, I just added path points in a cylinder up the body of the player.

Full Screen Effects



Lightning:

Full screen flash

Rain]

Noise texture streaked and filtered. Just randomly sample at different UVs in the texture. No need to actually simulate individual rain drops.

Fog Mist

Animated texture (like an inverse caustics one) stretched over the full screen panned slowly (same texture sequence as used in the front end).

Pan left and right when player moves left and right.

Cycle to the next texture in sequence as you move forward or back ward.

Conclusion

- Consider the end goal and don't get blinded by paths to get there.
- Faith Vision and Cleverness go a long way.
- You can probably figure a way to pull it off... or something that's really close.