Camera Transitions

By: Ángel González

What is a Camera Transition?

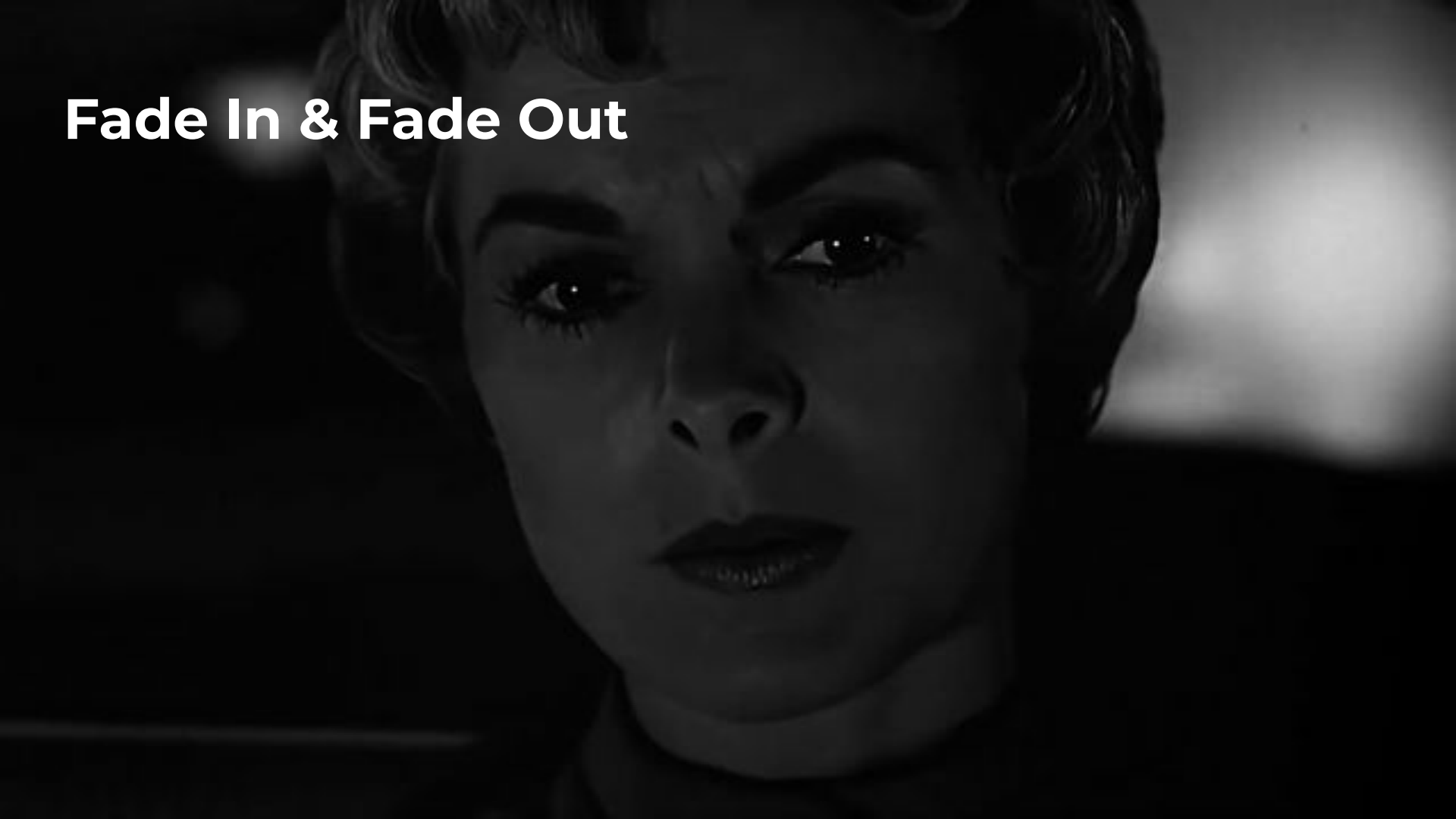
“Technique used in the post-production process of film editing and video editing by which scenes or shots are combined. Most commonly this is through a normal cut to the next shot.”

Wikipedia

Basic Transition Examples

Cut

Fade In & Fade Out



Wipe

Dissolve

Zoom

Why apply camera transitions to video games?


STEAM
PlayStation

The reasoning behind camera transitions

- We need time to load/unload a scene.
- This load/unload time will freeze our game.
- Implemented to hide the change process and the “freeze” time.

Other uses for Video Game Transitions

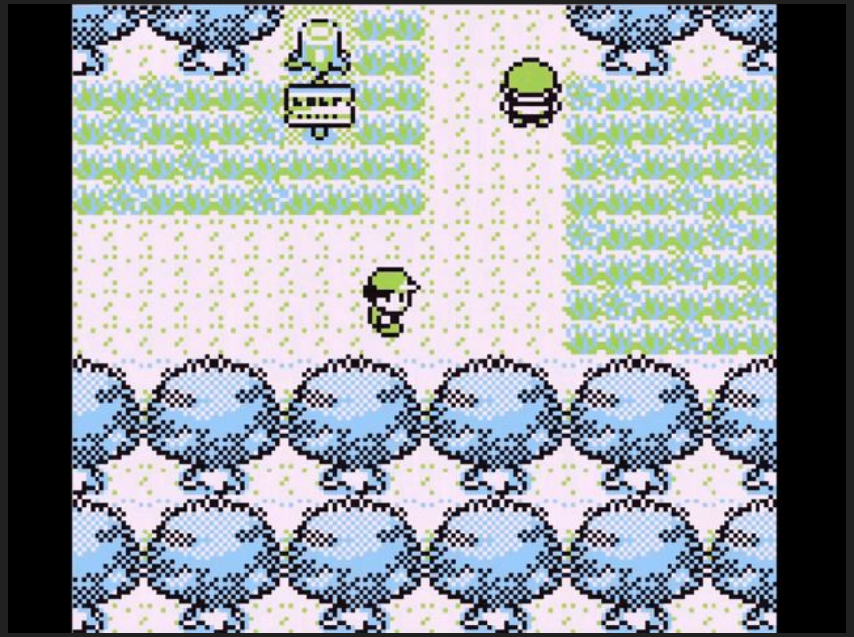
- Move forward or backward in time.
- To spice up the narrative or the cutscenes.
- *AESTHETICS*


STOP THIS.
STOP THIS NOW.

Video Game Transition Techniques

Manipulating geometrical forms

- Just “moving” geometrical forms around.
- Can have many shapes but are limited behaviour wise.
- Really simple code wise.

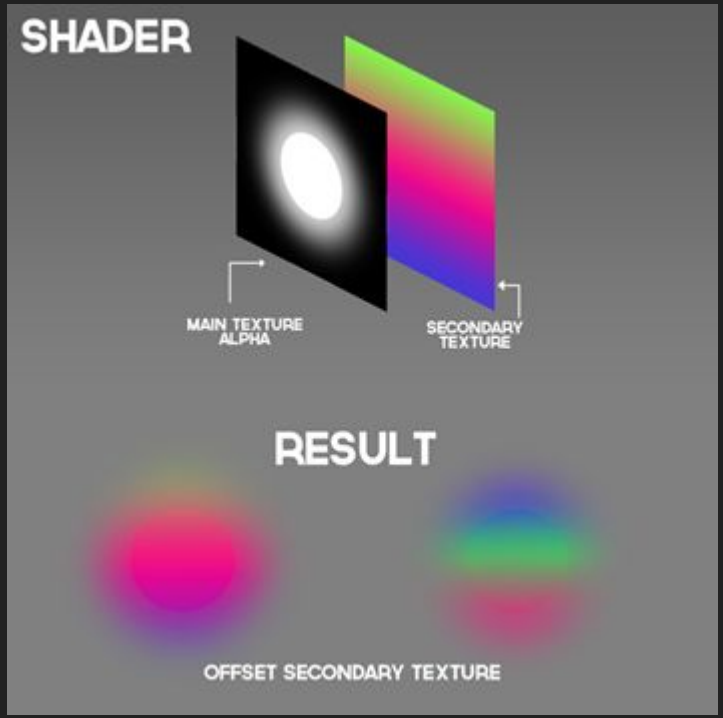


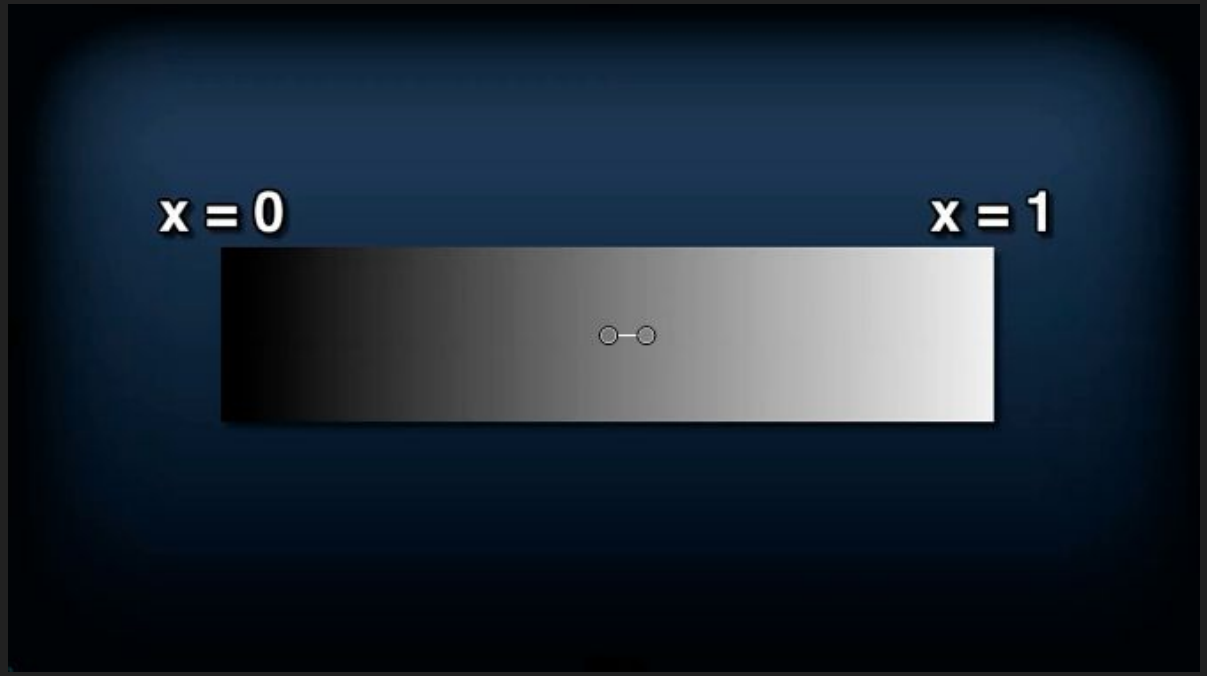
Manipulating the scene texture

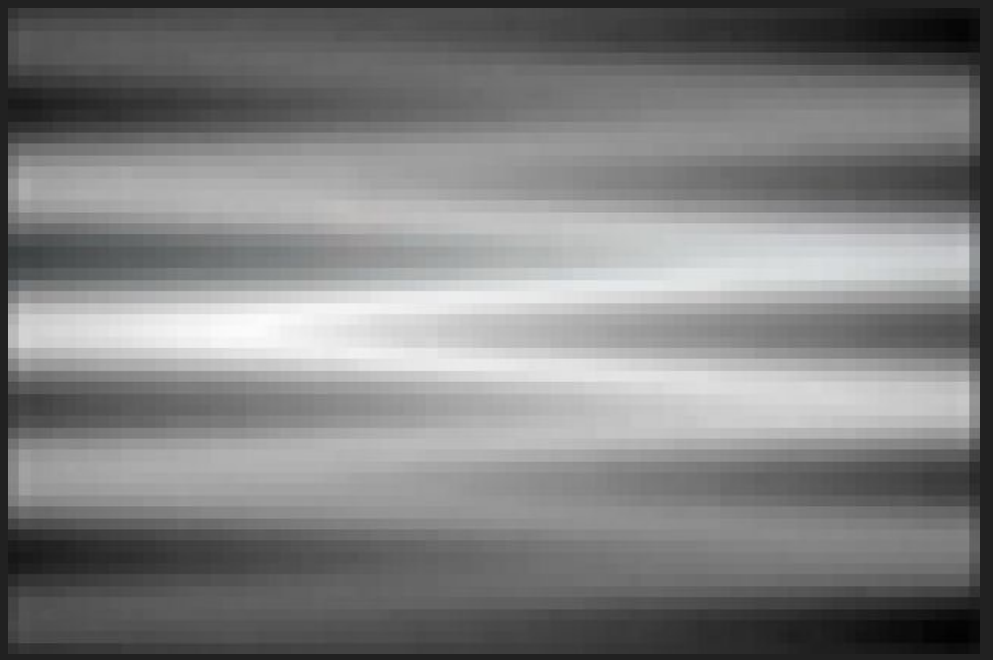
- All the textures are blended together and manipulated.
- Really flexible.
- Code wise its quite complex to make transitions like the one in the example.

Shaders

- Chunks of code that are executed in the GPU
- They modify the way an image is rendered.
- Using shaders to make transitions is a widespread practice due to the possibilities they bring.




x = 0
x = 1



If you want to know more about using shaders to create transitions, check the video linked below.

It will make for a great starting point.

Shaders Case Study - Pokémon Battle Transitions

Video Game Transition Examples

Disclaimer:

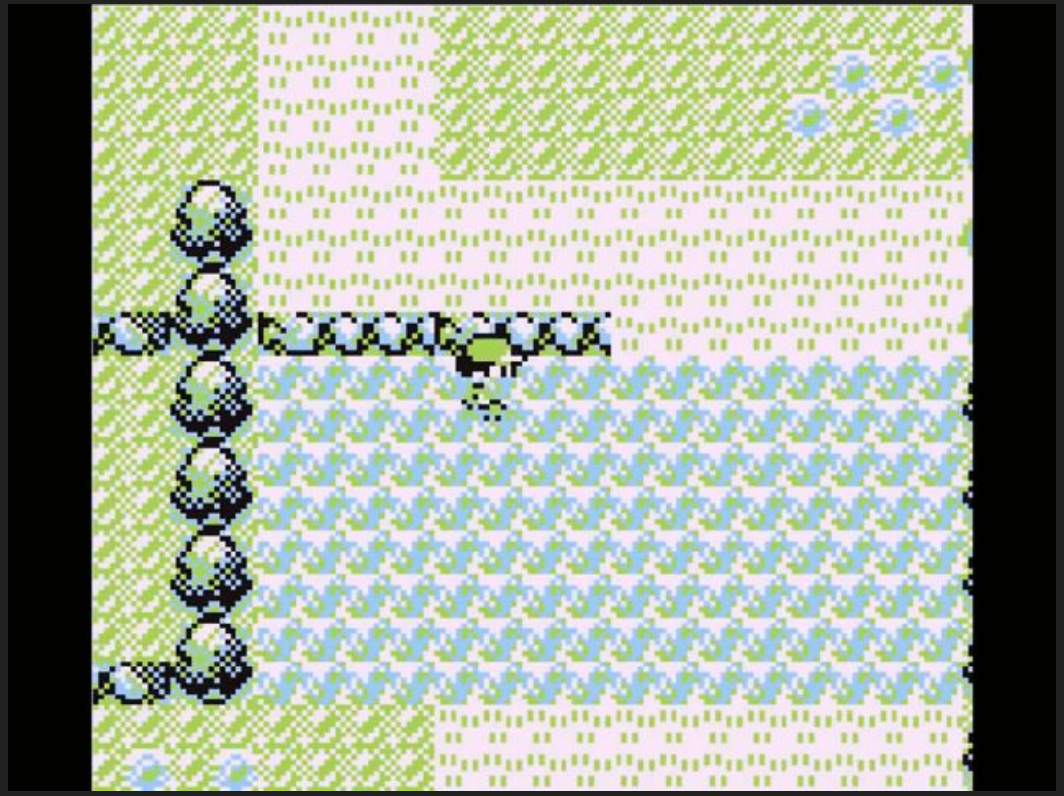
The majority of the transitions that will be set as examples come from Pokemon games and games with battle transitions.

Common Transitions

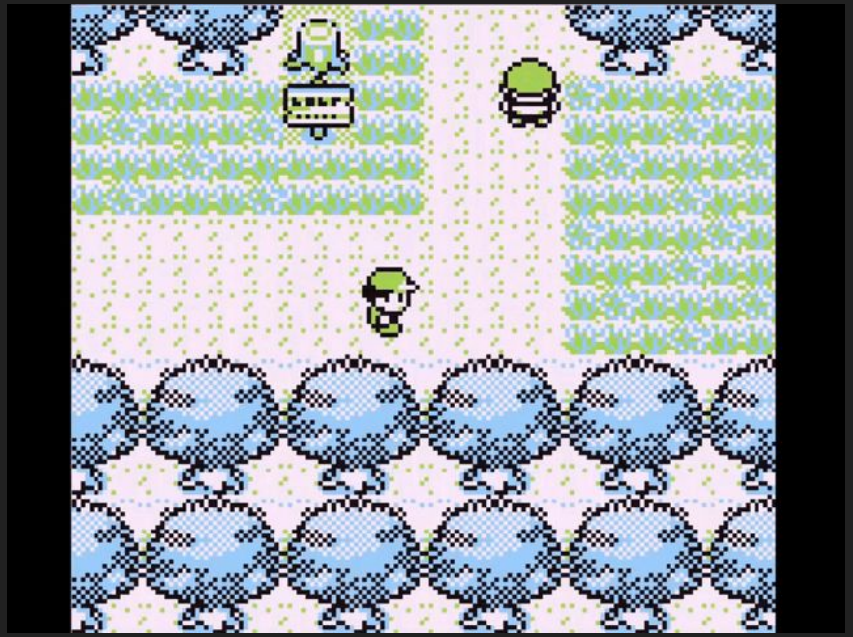
Fade To Black

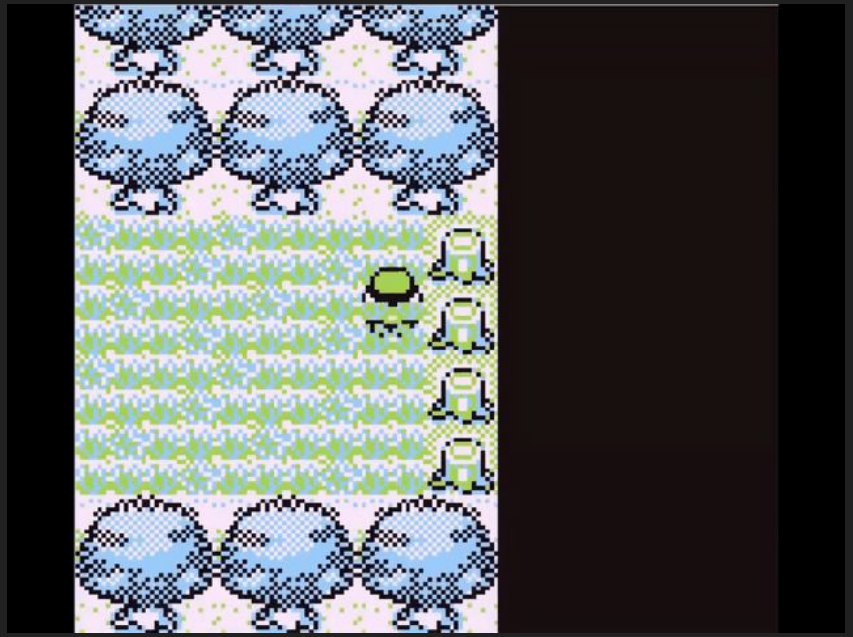
Fancy Fade To Black

Wipe

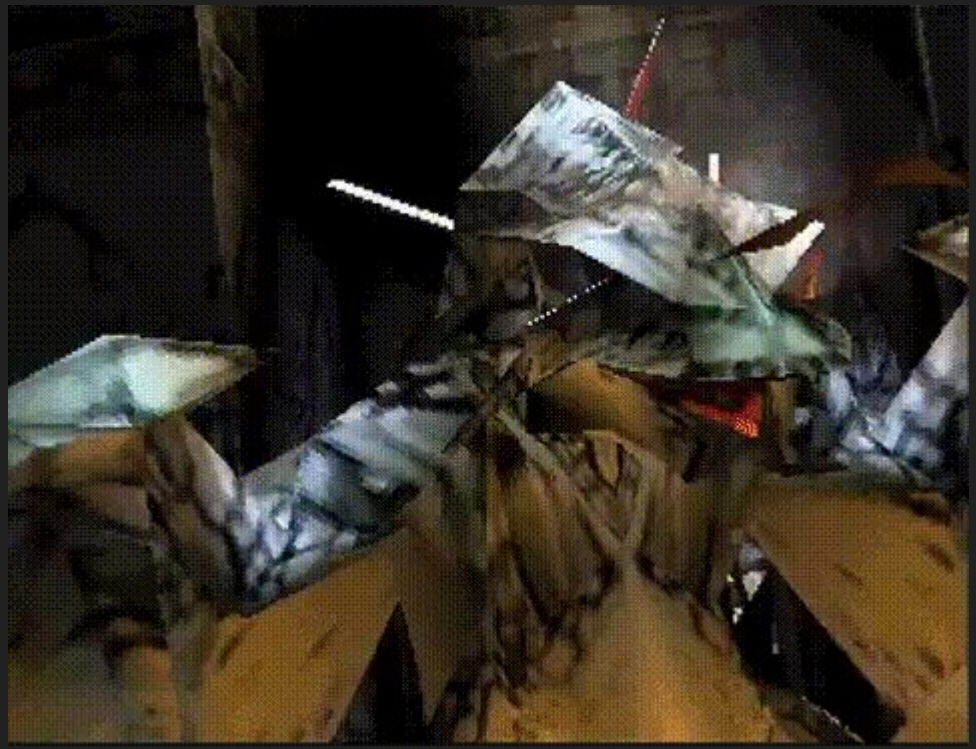


Horizontal & Vertical Alternating Bars





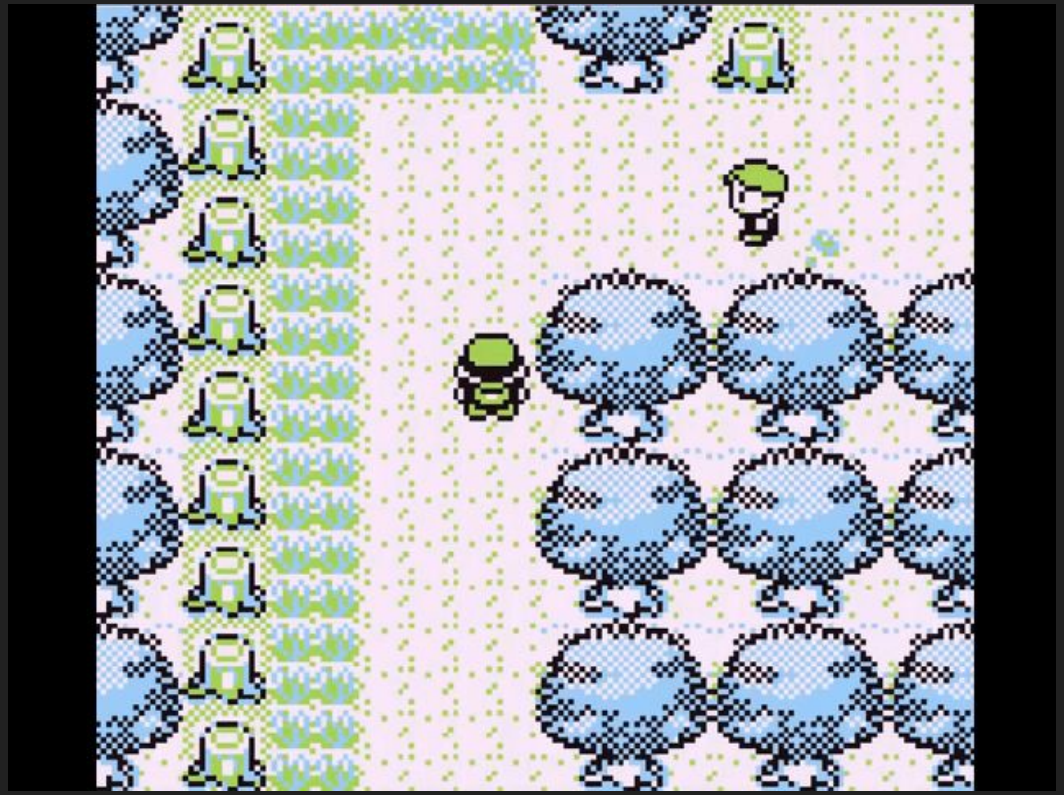
Camera Translation



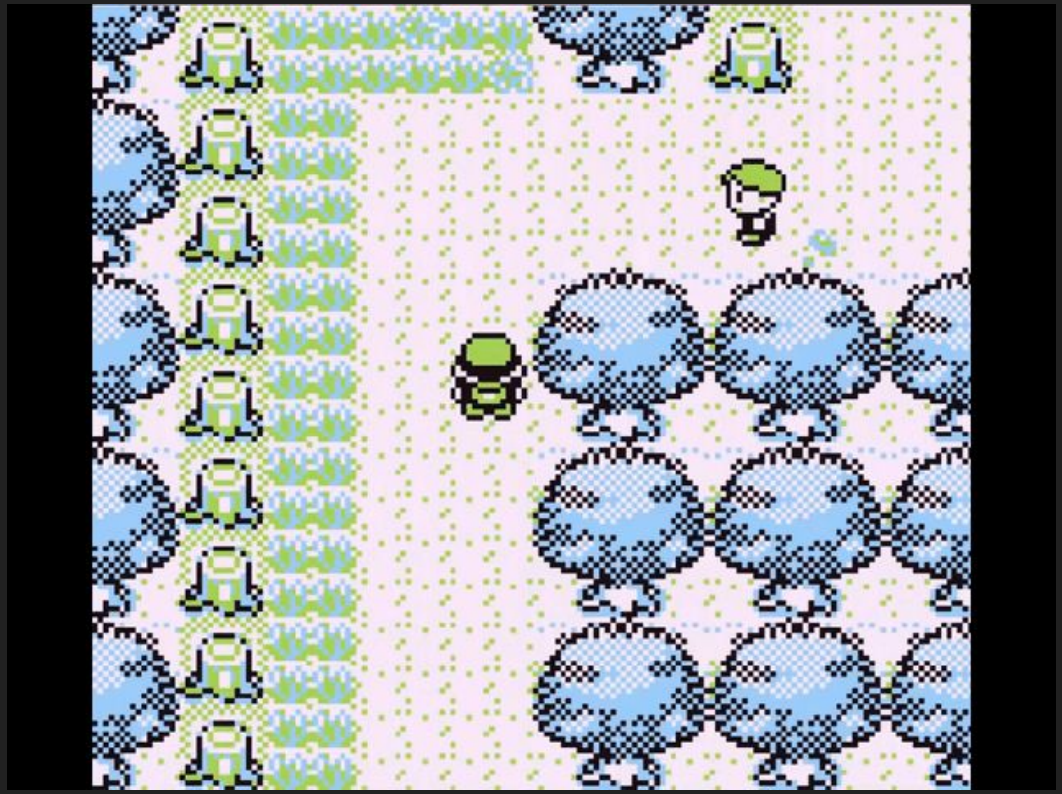
Composition of basics

Advanced Transitions

Sliding Quarters



Texture re-scaling



Texture manipulation (Flakes / Scales)

Texture manipulation (Blender effect)

Composition of advanced

Themed Transitions

Pokemon: Battle against a Gym Leader Brock

Pokemon: Battle against Zygarde

Resident Evil: Door Transition

Multiple games: Mirror Break Effect

Code Implementation

System Implementation

There are 4 core elements in this system:

- The Scene Manager: Creates/Updates/Changes the current_scene.
- The Scene Class: Base class of all scenes.
- The Transition Manager: Creates/Updates/Destroys the active_transition.
- The Transition Class: Applies the transition effect and calls the Scene Manager method SwitchScene() halfway through the transition.

Scene Manager

```
class SceneManager : public Module
{
public:
    SceneManager();
    ~SceneManager();

    bool Awake(pugi::xml_node& config);
    bool Start();
    bool PreUpdate();
    bool Update(float dt);
    bool PostUpdate();
    bool CleanUp();


public:
    void    ScenePushbacks();
    void    LoadInitialScene();

    void    SwitchScene(SCENES scene_name);
    void    LoadScene(SCENES scene_name);
    void    UnloadScene(Scene* scene_to_unload);

    Scene*  CreateScene(SCENES scene_name);

public:
    Scene*                  current_scene;
    Scene*                  next_scene;

private:
    std::vector<Scene*>     scenes;
};
```

Scene Class

```cpp
enum class SCENES
{
    FIRST_SCENE,
    SECOND_SCENE,
    NONE
};

class Scene
{
public:
    Scene(SCENES scene_name);
    virtual ~Scene();

    virtual bool Awake(pugi::xml_node& config);
    virtual bool Start();
    virtual bool PreUpdate();
    virtual bool Update(float dt);
    virtual bool PostUpdate();
    virtual bool CleanUp();

public:
    virtual void InitScene();
    virtual void DrawScene();

    virtual void ExecuteTransition();
    virtual void CameraDebugMovement(float dt);

public:
    SCENES      scene_name;

    int         map_width;
    int         map_height;
};
```

Transition Manager

```
class TransitionManager :  public Module
{
public:
    TransitionManager();
    ~TransitionManager();

    bool PostUpdate();

    bool CleanUp();

public:
    void DeleteActiveTransition();

    Transition* CreateCut(SCENES next_scene);

    //
    // And all other create methods.
    //

public:
    bool                        is_transitioning;

private:
    Transition*                 active_transition;

};
```

Transition Class

```
enum class TRANSITION_STEP
{
    NONE,
    ENTERING,
    CHANGING,
    EXITING
};

class Transition
{
public:
    Transition(SCENES next_scene, float step_duration, bool non_lerp = false);
    virtual ~Transition();

    virtual void Start();
    virtual void StepTransition();
    virtual void CleanUp();

public:
    virtual void Entering();
    virtual void Changing();
    virtual void Exiting();

    float Lerp(float start, float end, float rate);
    float N_Lerp(float start, float end, float rate, bool smash_in = false);
    float GetCutoffRate(float step_duration, float dt = App->GetDT());

public:
    TRANSITION_STEP step;
    SCENES          next_scene;

    float           step_duration;
    float           current_cutoff;

    bool            non_lerp;

private:
    float           cutoff_rate;

};
```

Transition Structure

There are 3 steps in the transition structure:

- **Entering:** Step before switching scenes. Will start the transition.
- **Changing:** Step and frame at which the scene switch happens.
- **Exiting:** Step after scene switch. Will end and delete the transition.

Transition Structure

```
void FadeToColour::StepTransition()
{
    switch (step)
    {
    case TRANSITION_STEP::ENTERING:

        Entering();

        break;

    case TRANSITION_STEP::CHANGING:

        Changing();

        break;

    case TRANSITION_STEP::EXITING:

        Exiting();

        break;
    }

    ApplyTheTransitionEffect();
}
```

The math behind the transitions

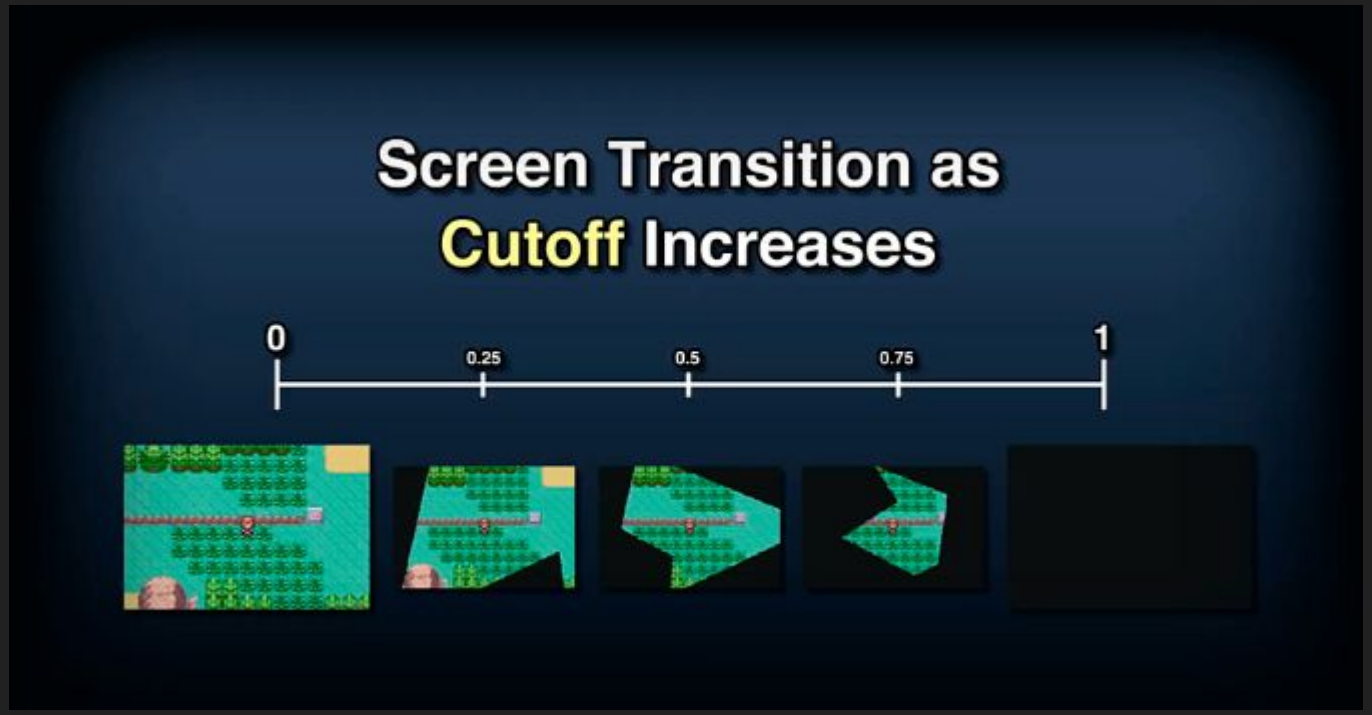
Cutoff Rate & Related Variables

```
float Transition::GetCutoffRate(float step_duration, float dt)
{
    cutoff_rate = dt / step_duration;

    return cutoff_rate;
}
```

- cutoff_rate is a constant value that will range from 0 to 1.
- current_cutoff will accumulate or subtract the cutoff_rate each loop.
- The transition step will change when the total value is either 0 or 1.

Relation Cutoff / Transition



Cutoff Example: Wipe

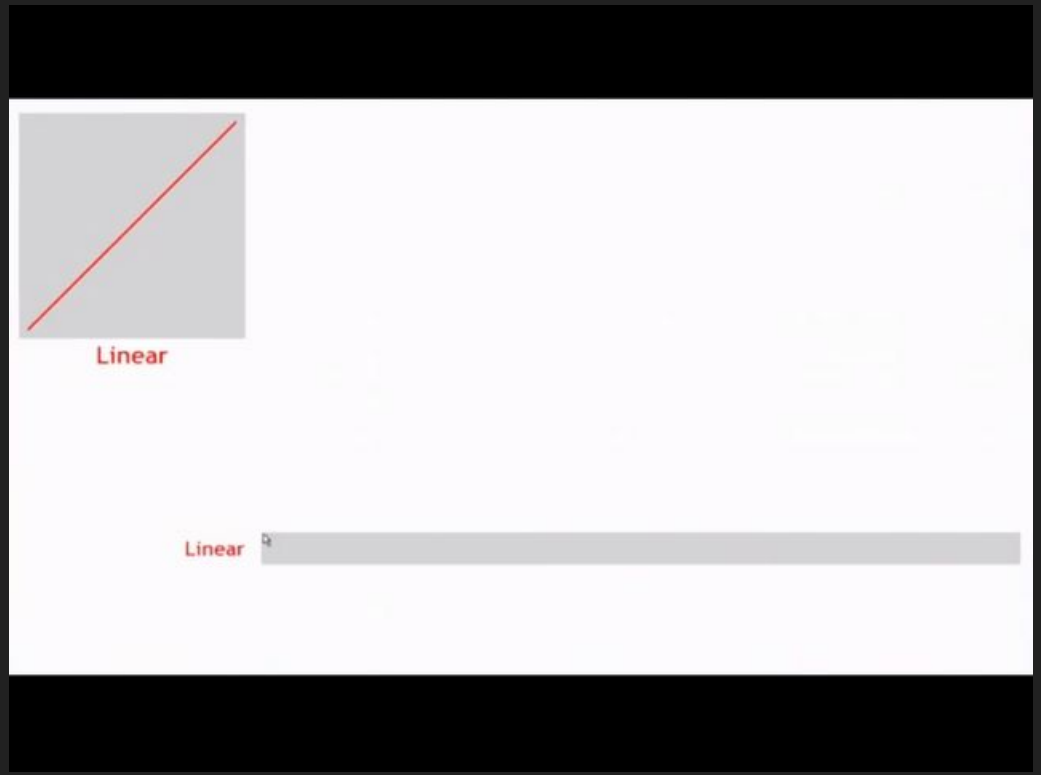
Lerp()

```
float Transition::Lerp(float start, float end, float rate)
{
    float increment = (end - start) * rate;

    return start + increment;
}
```

- Will linearly interpolate two given values.
- Rate goes from 0.0f (0 %) to 1.0f (100 %) advancement per frame.
- If Lerp(0.0f, 10.0f, 1.0f), then the advancement per frame will be of 10.0f.

Graphical Representation of Lerp()



N_Lerp()

```
float Transition::N_Lerp(float start, float end, float rate, bool smash_in)
{
    float increment = 0;

    if (smash_in)
    {
        increment = (end - start) * rate * rate;
    }
    else
    {
        float r = 1 - ((1 - rate) * (1 - rate));
        increment = (end - start) * r;
    }

    return start + increment;
}
```

Ease In and Ease Out N_Lerp()

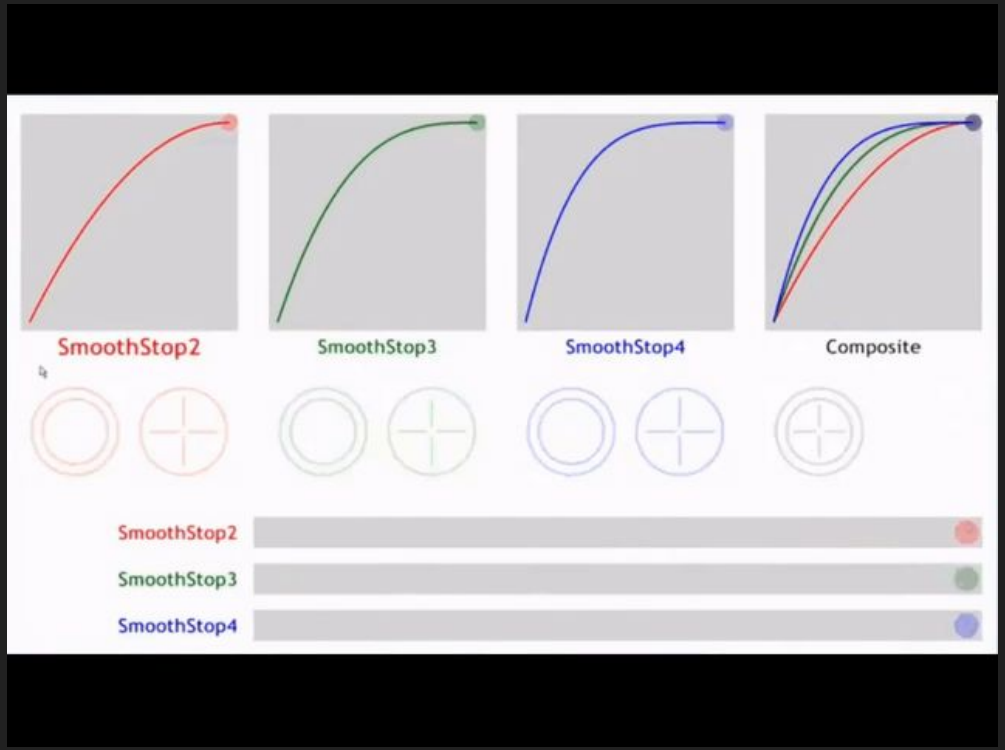
- Will non-linearly interpolate two given values.
- That aside these methods works the same as Lerp().
- The Ease In N_Lerp() will start slow end fast. (Ease In, Smash Out)
- The Ease Out N_Lerp() will start fast end slow. (Smash In, Ease Out)

Graphical Representation of Ease In N_Lerp()

- Starts Slow
- Ends Fast

Graphical Representation of Ease Out N_Lerp()

- Starts Fast
- Ends Slow



If you want to know more about interpolation methods your games check the video linked below.

Math for Game Programmers: Fast and Funky 1D Nonlinear Transformations

Let’s Make Some Transitions!

Disclaimer:

The intended result of the TODOs can be checked out at

exercises/solution/Game/solution.exe

TODO 0

```
// TODO 0: Familiarize yourself with the structure, different methods and core elements of
// the Camera Transition System. It will be key to be able to keep up with the rest of TODOs.
```

- Check the methods and variables of each of the core elements.
- Each important method has been commented, so it will be easy :)
- Make sure you understand everything.

TODO 1

```
// TODO 1: Switch the current scene for the next one. Remember that the transition needs to be deleted.
// Tip: Check SwitchScene() and DeleteActiveTransition().
```

- Check SwitchScene() & DeleteActiveTransition()
- To check that the transition works Press 1
- Should it not be done correctly, the transition will not loop.

TODO 2

```
// TODO 2: Initialize the variables of the FadeToColour transition.
// Initialize the screen rect with the parameters of the window's rect.
// After that, the  transition needs to get going. For that change the transition step.
// Tip: Remember to check StepTransition() and which elements a transition has.
```

- Check the variables of the FadeToColour transition.
- The window parameters should be taken into account.
- Remember that the step variable exists :)

TODO 3

```
// TODO 3: Add the cutoff_rate to the cutoff_rate each loop.
// Moreover, change the transition step when the current_cutoff reaches the MAX limit value.
// Tip: Check what current_cutoff and current_rate is.
```

- There is a function that return the cutoff_rate given a step_duration.
- The MAX limit value is already defined. Think about where :)

TODO 4

```
// TODO 4: Substract the cutoff_rate off of the current_cutoff each loop. Very similar to the previous TODO, but reversed.
// In addition to that, when the current_cutoff reaches the MIN limit value, change the step and delete the transition.
// Tip: Remember what you did in TODO 1.
```

- Very similar to the previous TODO.
- Check for MIN value insead for MAX value.
- Remember what you did in TODO 1.

TODO 5

```
// TODO 5: It's time to apply the current_cutoff to make the alpha fluctuate.
// Make use of the SDL_SetRenderDrawColor() and SDL_RenderFillRect() to draw
// the screen rect as well as to change it's alpha over time.
// Tip: Remember the value range of current_cutoff. Do not forget about fade_colour.
```

- Check SDL_SetRenderDrawColor() and SDL_FillRect().
- Take into account how current_rate fluctuates.
- Remember which is the value range of the alpha :)

TODO 6A

```
// TODO 6A: Code a Create method that creates FadeToColour transitions.
// Tip: See which parameters need to be passed to the FadeToColour transition.
```

- Check the variables that FadeToColour receives.

TODO 6B

```
// TODO 6B: Code the FadeToColour Create method definition.
// Tip: It's very similar to the other Create method definitions.
```

- The method should check if a transition is already executing.
- The new transition must be assigned to a variable. Think about which one :)
- The method should somehow notify that a transition has been loaded.

TODO 7A

```
// TODO 7A: Create and execute a FadeToColour transition on input. Remember in which scene you currently are.
// Tip: Actually it's only one line of code.
```

- Just call the Create method you coded in the previous TODOs :)
- Take into account to which scene do you want to go.

TODO 7B

```
// TODO 7B: Create and execute a FadeToColour transition on input. Remember in which scene you currently are.
// Tip: Actually it's only one line of code.
```

- Almost the same as the previous TODO.
- Again, take into account to which scene do you want to go.
- Experiment with the parameters of the method :)
- Press 2 to see whether or not the transition was correctly implemented.

TODO 8

```
// TODO 8: Make the camera move from it's origin position to the mouse's position.
// You will need to use either Lerp() or N_Lerp() as well as the next_pos buffer to move the camera accurately.
// Tip: Check what Lerp() and N_Lerp() do and which values they require as parameters.
```

- Make sure you understand what Lerp() and N_Lerp() do.
- Once you understand the methods, just pass the correct parameters :)

TODO 9A

```
// TODO 9A: Recreate the Slide effect by making use of either Lerp() or N_Lerp() like in the previous TODO.
// Use the screen rect variable. current_cutoff still behaves like in the previously seen transitions.
// Tip: Only one line of code is needed for each case.
```

- The screen rect should go from one edge to the other and return.
- Just apply what you have learned about Lerp() and N_Lerp() :)
- Again, remember how current_cutoff fluctuates.

TODO 9B

```
// TODO 9B: Very similar to its horizontal counterpart counterpart
// Think about which parameter of the screen use to reproduce the effect vertically for the Lerp()//N_Lerp().
// Tip: Only one line of code is needed for each case.
```

- Very similar to the previous TODO, but the transition is vertical.
- Think about which screen parameter needs to be modified.
- Remember to pass the correct parameters to Lerp() or N_Lerp().

TODO 10A

```
// TODO 10A: Recreate the wipe effect by making the screen rect cross from either of both sides to the other using Lerp() or N_Lerp().
// To make the rect cross the screen, the transition needs to be broken down into two steps using the transition steps.
// Tip: Be careful with current_step. It will range from 0 to 1 differently from other transitions.
```

- The screen rect should cross the screen from one side to the other.
- Again, the window parameters should be taken into account.
- This time, current_cutoff fluctuates differently, check out how :)
- Notice how the method is broken down in different steps. Think about why :)

TODO 10B

```
// TODO 10B: Again, similar to it's horizontal counterpart, but taking into account that
// the wipe will happen from top to bottom and from bottom to top.
// Now you have to code the step breakdown yourself.
```

- Again, very similar to the previous TODO, but vertical.
- It’s your turn to code the step breakdown now :)

Think about how to adapt this system to your project

Thank you for your attention!

- Mail: angotov@gmail.com
- GitHub: BarcinoLechiguino