Camera Battle Transitions

Biel Rabasa Galan

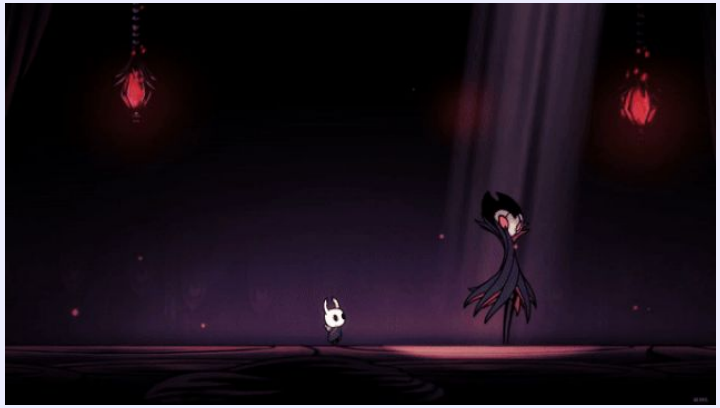
Transitions

Main purpose: hide abrupt cuts

Types:

- To color/image
- Scene combination

Examples in videogames



```
enum class TRANSITION_TYPE {
    NONE,
    FADE_TO_BLACK,
    SQUARED,
    CIRCLE,
    SLASH,
    THEATRE,
    DISSOLVE,
    ZOOM
};
```

```
class ModuleTransitions : public Module
{
public:
    ModuleTransitions(Application* app, bool start_enabled = false);
    ~ModuleTransitions();

    //Current scene, Next scene, transition time in frames, transition type
    void Transition(Module* preScene, Module* postScene, int transitionTime, TRANSITION_TYPE transitionType)
    {
        //The module is already transitioning
        if (transitionTime < 0) {
            LOG("ERROR: INVALID TRANSITION TIME");
            return;
        }

        //The module is already transitioning
        if (IsEnabled()) {
            LOG("ERROR: ALREADY TRANSITIONING");
            return;
        }

        Enable();
        this->preScene = preScene;
        this->postScene = postScene;
        this->transitionTime = transitionTime;
        this->transitionType = transitionType;
    }
```

Main function → **Transition(scene1, scene2, time, type)**

Scene Change: Changes between main scenes

Sprite: Stores the possible sprite

Step: Tracks the transition frames

Transition Functions

```
    bool Start();
    update_status Update();
    update_status PostUpdate();
    bool CleanUp();

private:

    //PreScene disable, PostScene enable
    void SceneChange();

    //TransitionModules
    void FadeToBlack();
    void Squared();
    void Circle();
    void Slash();
    void Theatre();
    void Dissolve();
    void Zoom();

    //Transition Sprites
    SDL_Texture* sprite = nullptr;

    //Variables
    int step;
    int transitionTime;
    TRANSITION_TYPE transitionType;
    Module* preScene;
    Module* postScene;
};
```

```
void ModuleTransitions::SceneChange()
{
    preScene->Disable();
    postScene->Enable();
}
```

Scene Change: Changes scenes

CleanUp: Erase screenshot

CleanUp: Reset texture blend mode

```
bool ModuleTransitions::Start()
{
    bool ret = true;

    step = 0;

    return ret;
}

bool ModuleTransitions::CleanUp()
{
    preScene = nullptr;
    postScene = nullptr;

    //Sets blendmode to default + unload texture
    SDL_SetTextureBlendMode(sprite, SDL_BLENDMODE_NONE);
    App->textures->Unload(sprite);
    sprite = nullptr;

    //Erase screenshot if exists
    remove("Assets/screenshot.bmp");

    return true;
}
```

Start + CleanUp

```cpp
update_status ModuleTransitions::Update()
{
    step++;

    return UPDATE_CONTINUE;
}

update_status ModuleTransitions::PostUpdate()
{
    //FUNCTION EXECUTION SWITCH
    switch (transitionType) {
    case TRANSITION_TYPE::FADE_TO_BLACK:
        FadeToBlack();
        break;

    case TRANSITION_TYPE::SQUARED:
        Squared();
        break;

    case TRANSITION_TYPE::CIRCLE:
        Circle();
        break;

    case TRANSITION_TYPE::SLASH:
        Slash();
        break;

    case TRANSITION_TYPE::THEATRE:
        Theatre();
        break;
```

PostUpdate: Function Call

SceneChange: Specify moment

Transition Disable:

Disable itself

```cpp
    case TRANSITION_TYPE::DISSOLVE:
        Dissolve();
        break;

    case TRANSITION_TYPE::ZOOM:
        Zoom();
        break;
    }

    //CHANGING SCENE SWITCH
    switch (transitionType) {
    case TRANSITION_TYPE::ZOOM:
    case TRANSITION_TYPE::DISSOLVE:
        if (step >= 1)
            SceneChange();
        break;

    default:
        if (2 * step >= transitionTime)
            SceneChange();
        break;
    }

    if (step >= transitionTime)
        Disable();

    return UPDATE_CONTINUE;
}
```

Function calling + Scene changing

Fade to black


```
void ModuleTransitions::FadeToBlack()
{
    if (2 * step < transitionTime) {
        App->renderer->DrawQuad(SDL_Rect{ 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT }, 0, 0, 0,
            (float(step * 2) / (float)transitionTime) * 255.0f);
    }
    else if (step * 2 == transitionTime) {
        App->renderer->DrawQuad(SDL_Rect{ 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT }, 0, 0, 0, 255.0f);
    }
    else {
        App->renderer->DrawQuad(SDL_Rect{ 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT }, 0, 0, 0,
            255.0f - (float(step * 2) / (float)transitionTime) * 255.0f);
    }
}
```

Squared


```
void ModuleTransitions::Squared()
{
    int percentage = ((float)step / (float)transitionTime) * 40.0f;
    int w = SCREEN_WIDTH / 10;
    int h = SCREEN_HEIGHT / 10;

    if (step * 2 >= transitionTime)
        percentage = 40 - percentage;

    for (int j = 0; j < percentage; ++j) {
        for (int i = 0; i < percentage; ++i) {
            App->renderer->DrawQuad(SDL_Rect{ w * i, h * (10 - j + i), w, h }, 0, 0, 0, 255);
        }
    }
}
```

Circle


```
void ModuleTransitions::Circle()
{
    if (sprite == nullptr)
        sprite = App->textures->Load("Assets/circle.png");

    float percentage = ((float)step / (float)transitionTime) * 2.0f;

    if (step * 2 >= transitionTime)
        percentage = 2.0f - percentage;

    float scale = 5.0f * percentage;

    App->renderer->DrawTexture(sprite, SCREEN_WIDTH / 2.0f - 150.0f * scale, SCREEN_HEIGHT / 2.0f - 150.0f * scale, scale);
}
```

Slash


```
void ModuleTransitions::Slash()
{
    if (sprite == nullptr)
        sprite = App->textures->Load("Assets/slash.png");

    float percentage = ((float)step / (float)transitionTime) * 200.0f;

    if (step * 2 >= transitionTime)
        percentage = 200 - percentage;

    int scale = 3;

    App->renderer->DrawTexture(sprite, ((SCREEN_WIDTH / 2) * ((100.0f - percentage) / 100.0f)) + ((SCREEN_WIDTH - (600 * scale)) / 2),
        (SCREEN_HEIGHT - (300 * scale)) / 2, scale);

    App->renderer->DrawTexture(sprite, ((SCREEN_WIDTH - (600 * scale)) / 2) - ((SCREEN_WIDTH / 2) * ((100.0f - percentage) / 100.0f)),
        (SCREEN_HEIGHT - (300 * scale)) / 2, scale, true);
}
```

Theatre


```
void ModuleTransitions::Theatre()
{
    if (sprite == nullptr)
        sprite = App->textures->Load("Assets/semicircle.png");

    float percentage = ((float)step / (float)transitionTime) * 360.0f;

    int scale = 7;

    App->renderer->DrawTexture(sprite, ((SCREEN_WIDTH - (300 * scale)) / 2), SCREEN_HEIGHT, scale,
        false, NULL, 1.0f, percentage, 150 * scale, 0);
}
```

Screenshot

```cpp
update_status ModuleRender::PostUpdate()
{
    SDL_RenderPresent(renderer);

    if (screenshot)
        screenshot = false;

    if (pendingToScreenshot) {
        const Uint32 formats = SDL_PIXELFORMAT_ARGB8888;

        SDL_Surface* screen = App->window->screen_surface;
        SDL_RenderReadPixels(renderer, NULL, formats, screen->pixels, screen->pitch);
        SDL_SaveBMP(screen, "Assets/screenshot.bmp");
        SDL_FreeSurface(screen);

        pendingToScreenshot = false;
        screenshot = true;
    }

    return UPDATE_CONTINUE;
}
```

Dissolve

```
void ModuleTransitions::Dissolve()
{
    //If screenshot is made, load it
    if (App->renderer->screenshot) {
        sprite = App->textures->Load("Assets/screenshot.bmp");
    }

    //Call for screenshot
    if (sprite == nullptr) {
        App->renderer->pendingToScreenshot = true;
        return;
    }

    float percentage = ((float)step / (float)transitionTime) * 255.0f;

    SDL_SetTextureBlendMode(sprite, SDL_BLENDMODE_BLEND);
    SDL_SetTextureAlphaMod(sprite, 255.0f - percentage);

    App->renderer->DrawTexture(sprite, 0, 0);
}
```

Zoom

```
void ModuleTransitions::Zoom()
{
    //If screenshot is made, load it
    if (App->renderer->screenshot) {
        sprite = App->textures->Load("Assets/screenshot.bmp");
    }

    //Call for screenshot
    if (sprite == nullptr) {
        App->renderer->pendingToScreenshot = true;
        return;
    }

    float percentage = ((float)step / (float)transitionTime) * 5.0f;

    App->renderer->DrawTexture(sprite, 0, 0, 1.0f + percentage);

}
```

Your turn...

```
enum class TRANSITION_TYPE {
    NONE,
    FADE_TO_BLACK,
    SQUARED,
    CIRCLE,
    SLASH,
    THEATRE,
    DISSOLVE,
    ZOOM,

    //TODO 1: Crea un nou tipus de transisió
    CUT
    //

};
```

```
//TODO 2A: Executa la funció "Transition" de l'escena 1 a la 2, dona-li un temps de transisió
else if (App->input->GetKey(SDL_SCANCODE_SPACE) == KEY_DOWN)
    App->transitions->Transition(App->scene1, App->scene2, 40, TRANSITION_TYPE::CUT);
//
```

```
//TODO 3: Declara la funció per transicionar.
void Cut();
//
```

```
//TODO 4: Executa la funció de la transisió
case TRANSITION_TYPE::CUT:
    Cut();
    break;
//
```

```
switch (transitionType) {

//TODO 5: Assigna el tipus de transisió perquè faci el canvi de mòdul al principi.
case TRANSITION_TYPE::CUT:
//

case TRANSITION_TYPE::ZOOM:
case TRANSITION_TYPE::DISSOLVE:
    if (step >= 1)
        SceneChange();
    break;

default:
    if (2 * step >= transitionTime)
        SceneChange();
    break;
}
```

```cpp
//TODO 6: Defineix la funció de transició
void ModuleTransitions::Cut() {
    //If screenshot is made, load it
    if (App->renderer->screenshot) {
        sprite = App->textures->Load("Assets/screenshot.bmp");
    }

    //Call for screenshot
    if (sprite == nullptr) {
        App->renderer->pendingToScreenshot = true;
        return;
    }

    float percentage = ((float)step / (float)transitionTime) * ((float)SCREEN_HEIGHT / 2.0f);

    SDL_Rect top = { 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT / 2 };
    SDL_Rect bot = { 0, SCREEN_HEIGHT / 2, SCREEN_WIDTH, SCREEN_HEIGHT / 2 };
    App->renderer->DrawTexture(sprite, 0, -percentage, 1.0f, false, &top);
    App->renderer->DrawTexture(sprite, 0, SCREEN_HEIGHT / 2 + percentage, 1.0f, false, &bot);
}
//
```