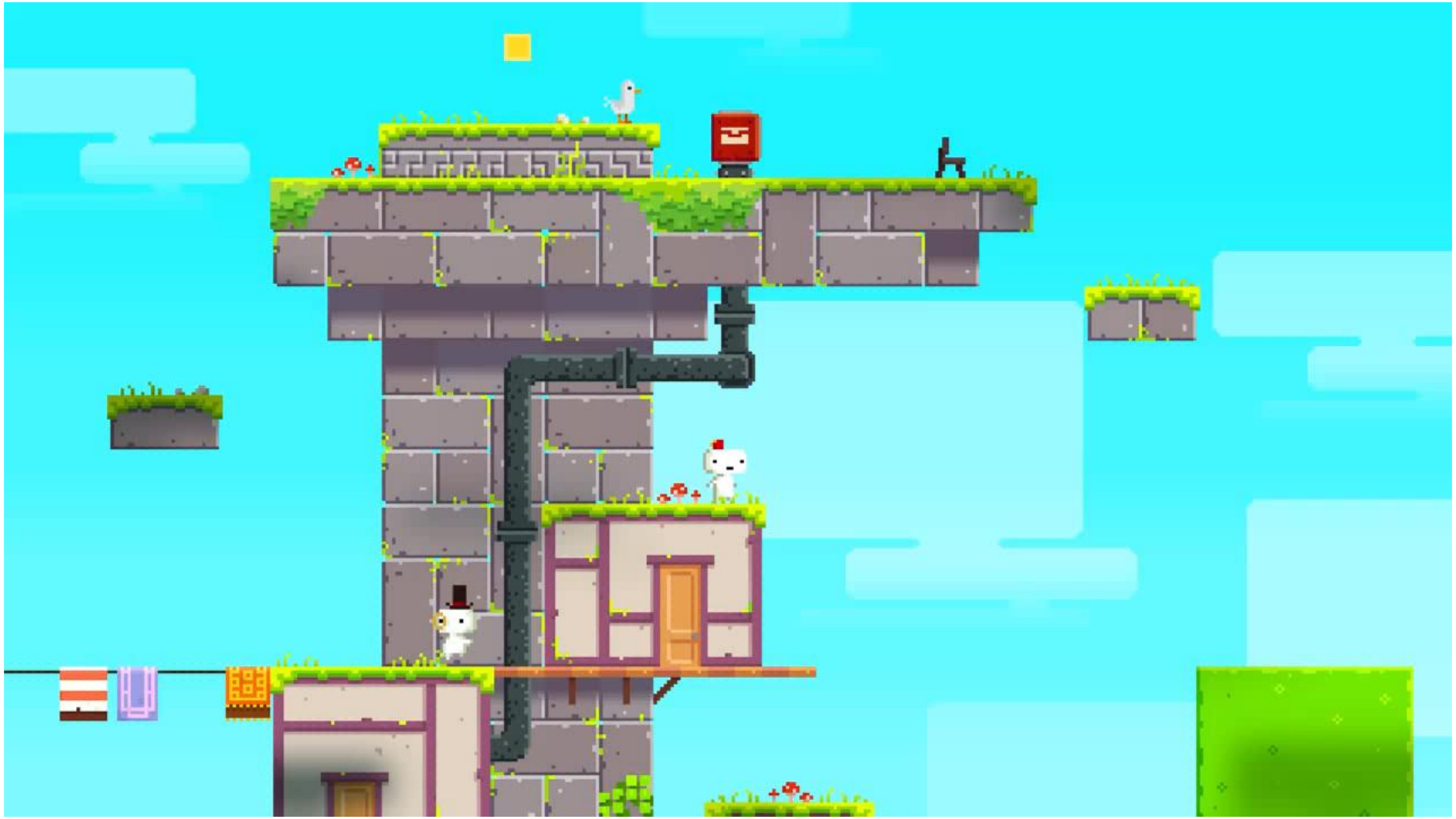
The Challenge of Bringing FEZ to PlayStation Platforms

Miguel Angel Horna
Lead Programmer at BlitWorks

GDC 'Eu

GAME DEVELOPERS CONFERENCE™ EUROPE
CONGRESS-CENTRUM OST KOELNMESSE · COLOGNE, GERMANY
AUGUST 11-13, 2014 · EXPO: AUGUST 11-12, 2014

polytron



FEZ

- Released in 2012
- Developed by Polytron Corporation
- In development for over 5 years
- Just 2 people
  - Game design/Art: Phil Fish
  - Game programmer: Renaud Bédard

FEZ

- C# Code
- Xbox 360 version uses XNA
- PC Version uses MonoGame (OpenGL)
- No C# runtime for PlayStation platforms

The main decision

- Two options:
  - Port C# to PlayStation platforms
    - Straightforward (almost) game code port
    - No JIT support. Doubts on performance
    - OpenGL to native libraries
  - Rewrite C# code to C++
    - “Native” performance. Better optimizations
    - Lots of work. Over 500 classes

The main decision

- We chose game code rewrite
- We knew it was going to be hard
- And long… It took almost 1 year to complete
- Manual rewrite was out of question
- Tried some automatic conversion tools

The main decision

- C# to C++ Converter by Tangible Software
- It made the first (rough) conversion
- Every file had to be completed, edited and reviewed
- Blind conversion. Much time passed with no visible progress (not even compiling)

The main decision

C#

```
public IEnumerableTrile> ActorTriles(ActorType type)
{
    return TrileSet == null ? Enumerable.Repeat<Trile>(null, 1) : TrileSet.Triles.Values.Where(x =>
x.ActorSettings.Type == type);
}
```

Auto converted

```
std::shared_ptr<IEnumerable<Trile*>> LevelManager::ActorTriles(ActorType type)
{
//C# TO C++ CONVERTER TODO TASK: Lambda expressions and anonymous methods are not converted to native
C++:
    return get_TrileSet() == nullptr ? Enumerable::Repeat<Trile*>(nullptr, 1) : get_TrileSet()-
>get_Triles().Values->Where(x => x.ActorSettings->Type == type);
}
```

The main decision

Final code

```
std::shared_ptr<List<std::shared_ptr<Trile>>> LevelManager::ActorTriles(ActorType type)
{
    if(get_TrileSet()==nullptr)
    {
        std::shared_ptr<List<std::shared_ptr<Trile>>> result = std::make_shared<List<std::shared_ptr<Trile>>>();
        result->push_back(nullptr);
        return result;
    }
    else
    {
        return Where(get_TrileSet()->get_Triles(),[=] (const std::shared_ptr<Trile> &t) { return t->get_ActorSettings()-
>get_Type() == type; } );
    }
}
```

Conversion

- Properties
  - Generate get_, set_ accessors in classes
  - Converter generated proper methods
  - .. But sometimes didn’t use them
  - Needed long rewrites
  - Beware of returning temporary references, some compilers don’t detect that!

Conversion

- Be careful with operation order with -= and too “automated” rewrite

C#
```
PlayerManager.Velocity -= destination - instance.Center;
```

C++ wrong
```
get_PlayerManager()->set_Velocity(get_PlayerManager()->get_Velocity() - destination - instance->get_Center());
```

C++ correct
```
get_PlayerManager()->set_Velocity(get_PlayerManager()->get_Velocity() - (destination - instance->get_Center()));
```

Conversion

- Extension methods
  - Extension methods were widely used ...
  - … on Enums
  - No way to convert it to C++
  - Just create normal functions passing the enum as parameter

C#
```
var bitangent = orientation.GetBitangent().AsAxis().GetMask();
```

C++
```
auto bitangent = FezMath::GetMask(FezMath::AsAxis(FezMath::GetBitangent(orientation)));
```

Conversion

- Lambda expressions & delegates
  - Game code heavily used lambda expressions for events, list queries ...
  - Delegates were used a lot too (threads, scheduled operations, scripting…)
  - They don’t have direct conversion to C++ …
  - … but they have for C++11
  - Lambdas and std::function<>

Conversion

C#
```
GameState.LoadSaveFile(() => GameState.LoadLevelAsync(Util.NullAction));
```

C++
```
get_GameState()->LoadSaveFile([=] ()
{
    get_GameState()->LoadLevelAsync([=] () { });
});
```

C#
```
return new LongRunningAction((elapsed, since) => component.IsDisposed);
```

C++
```
return LongRunningAction::Create([=] (float elapsed, float since) -> bool
{
    return component->get_IsDisposed();
});
```

Conversion

- Nested lambdas caused problems with the converter
- Corrupted, half-converted files
    - Unbalanced brackets generation
    - Unconverted c# code
    - Missing pieces of functions
- Find the offending code, comment, re-convert the file, and manually convert that piece of code

Conversion

C#

```
DotService.Say("DOT_ANTI_A", true, false).Ended = () =>
{ DotService.Say("DOT_ANTI_B", true, false).Ended = () =>
{ DotService.Say("DOT_ANTI_C", true, false).Ended = () =>
{ DotService.Say("DOT_ANTI_D", true, true).Ended = CheckCubes; };};};
```

C++

```
get_DotService()->Say("DOT_ANTI_A", true, false)->Ended = [=] ()
{
     get_DotService()->Say("DOT_ANTI_B", true, false)->Ended = [=] ()
     {
           get_DotService()->Say("DOT_ANTI_C", true, false)->Ended = [=] ()
           {
                 get_DotService()->Say("DOT_ANTI_D", true, true)->Ended = [=] { CheckCubes(); };
           };
     };
};
```

Garbage Collector

- FEZ had random stuttering due to the GC kicking in
- Deterministic object destruction preferred
- Reference counting
- C++11’s std::shared_ptr<>
- Caused its own kind of bugs ☹

Garbage Collector

- Circular references
- Try to access shared_from_this() in constructors
- Capturing **this** on lambdas caused problems (no reference increment)

Garbage Collector

```
void SoundEmitter::FadeOutAndDie(float forSeconds)
{
    Waiters::Interpolate(forSeconds, [=] (float s)
    {
        set_VolumeFactor(volumeFactor * (1 - s));
    }
    , [=] ()
    {
    if (get_Cue() != nullptr && !get_Cue()->get_IsDisposed() && get_Cue()->get_State() != SoundState::Stopped)
        get_Cue()->Stop(true);
    });
}
void MovingGroupsHost::MovingGroupState::StopSound()
{
    eAssociatedSound->FadeOutAndDie(0.25f);
    eAssociatedSound.reset();
}
```

Garbage Collector

```
void SoundEmitter::FadeOutAndDie(float forSeconds)
{
    auto _this=shared_from_this();

    Waiters::Interpolate(forSeconds, [=] (float s)
    {
        _this->set_VolumeFactor(volumeFactor * (1 - s));
    }
    , [=] ()
    {
    if (_this->get_Cue() != nullptr && !_this->get_Cue()->get_IsDisposed() && _this->get_Cue()->get_State() != SoundState::Stopped)
        _this->get_Cue()->Stop(true);
    });
}
```

Reflection

- Game scripting used reflection to:
  - Trap scripting objects events by name
  - Invoke methods by name
- We implemented method, event, triggers… descriptors for scripting objects
- Generated dynamic methods by using IL Emit
- No dynamic code generation

Reflection

```
script key=9 {   name "Untitled"
        triggers {  trigger {   event "Enter"
              object { type "Volume"  identifier 4
        }  }  }
        actions {   action {
                    operation "ChangeLevelToVolume"
                    arguments "LIGHTHOUSE_HOUSE_A" "1" "True" "True"
                    object {
                       type "Level"
                    }
        }  }  }
```

Reflection

```
  std::shared_ptr<EntityDescriptor> IVolumeService::GetEntityDescriptor()
  {
  //Operations
  std::unordered_map<String,MethodDesc> operations;
  operations["FocusCamera"] = MethodDesc([=] (std::vector<MultiTypeParameter> &params) -> std::shared_ptr<LongRunningAction>
{return FocusCamera(params[0].I, params[1].I, params[2].B); },3 ) ;
  operations["SetEnabled"] = MethodDesc([=] (std::vector<MultiTypeParameter> &params) -> std::shared_ptr<LongRunningAction>
{SetEnabled(params[0].I, params[1].B, params[2].B); return nullptr;},3 ) ;
  //Properties
  std::unordered_map<String,PropertyDesc> properties;
  properties["GomezInside"] = PropertyDesc([=] (int id) -> MultiTypeParameter { MultiTypeParameter mt; mt.B=get_GomezInside(id);
return mt; },MultiType::BOOL);
  //Events
  std::unordered_map<String,EventHandlerDesc> events;
  events["Enter"] = EventInstance(&Enter,&Exit);
  events["Exit"] = EventInstance(&Exit);

  return std::make_shared<EntityDescriptor>("Volume","Volume",false,operations,properties,events);
  }
```

Reflection

- Service dependencies were resolved using reflection

```
 [ServiceDependency]
public ISoundManager SoundManager { protected get; set; }
[ServiceDependency]
public IContentManagerProvider CMProvider { protected get; set; }
```

- In C++ there are no attributes, and also we can’t get the property names
- This was not possible to resolve the same way

Reflection

- We created a base class with all the service set_ methods virtual, and doing nothing
- The dependency resolver iterated all registered services
- A macro called for each combination of property name and type, tried to dynamic_cast the service to the type, and if so, called the set_ method
- All components derived from this class, so the overridden set_ functions actually set the value

Reflection

```
#define MY_SERVICE_SET(__propertyname,__serviceclass) \
   if(std::dynamic_pointer_cast<__serviceclass>(service)!=nullptr) \
   {\
         std::shared_ptr<__serviceclass>
   private_##__propertyname##_##__serviceclass=std::dynamic_pointer_cast<__serviceclass>(service);\
         injector->set_##__propertyname(private_##__propertyname##_##__serviceclass);\
   }



   MY_SERVICE_SET(CameraManager,IGameCameraManager);
   MY_SERVICE_SET(SoundManager,ISoundManager)
   MY_SERVICE_SET(CMProvider,IContentManagerProvider)
   MY_SERVICE_SET(TargetRenderer,ITargetRenderingManager)
   MY_SERVICE_SET(TargetRenderingManager,ITargetRenderingManager)
   ...
```

.NET Framework

- Replacement for some .NET Framework libraries:
  - Threads & Synchronization
    - Almost direct map to native APIs
    - lock() blocks converted to mutex.Lock() and Unlock()
  - Files
    - Also easy to map, and BinaryReader/Writer gave us the endian safe file access.

.NET Framework

- String (Unicode)
  - String class derived from std::wstring, but with C# like methods (SubString(),Replace(),Split()..)
- Collections (Dictionary, List, Array)
  - Converter automatically changed them to STL containers (unordered_map,list,vector)
- Nullable types
  - Implemented our own Nullable<T> class

.NET Framework

- Events
  - Internally containing a std::list<std::function>
  - Adding a method to an event requires a lambda
    - **C#** `CameraManager.ViewpointChanged += UpdateRotation;`
    - **C++** `get_CameraManager()->ViewpointChanged += [=] () {UpdateRotation();};`
  - Initially same interface than C# (+=, -=)
  - No way to compare lambdas to remove, so we had to change interface and return ID
    - `UpdateRotationDelegateId=get_CameraManager()->ViewpointChanged.Add([=] () {UpdateRotation();});`
    - `get_CameraManager()->ViewpointChanged.Remove(UpdateRotationDelegateId);`

.NET Framework

- LINQ was widely used to search
- We implemented LINQ-like operations for STL list,vector, map:
  - Where, Any, All, Union, Exists...
- Usage was a bit awkward sometimes

C#
```
if(!tracks.Any(y => y.Name == x.Track.Name))
```

C++
```
if(!Any<std::shared_ptr<AmbienceTrack>>(tracks,[=] (const std::shared_ptr<AmbienceTrack>
&y) -> bool { return y->get_Name()==(*x)->Track->get_Name(); }))
```

Bugfixing

- Original game already had bugs
- C# to C++ introduced new ones
  - Memory leaks (circular references)
  - Mixing normal and reference counted pointers to same object. Use after delete.
  - Returning reference to temporary object in get_ properties access:

```
const Vector3 &TrixelEmplacement::get_Position() const
{
    return Vector3(X, Y, Z);
}
```

Bugfixing

- Mistakes during manual conversion
  - Missing parenthesis on -= operations
  - Errors converting lambda expressions in LINQ operations
- Uninitialized member variables
- C++11 compiler bugs !!

Optimization

- OK, now it worked… but slowly
- CPU intensive code
- Lots of geometry with inefficient instancing
- Complex shaders

Optimization (CPU)

- std::shared_ptr is thread safe. We created a lightweight (unsafe) fast_ptr
- Move some CPU intensive operations to another thread
- Erasing std::vector elements is slow:
  - convert to std::list
  - replace removed element with last one and resize()

MonoGame graphics

- For PC, OpenGL based.
- Two options:
  - Minimal OpenGL library for each platform
  - Custom MonoGame target for each platform
- We anticipated GPU performance problems
- Chose custom MonoGame targets

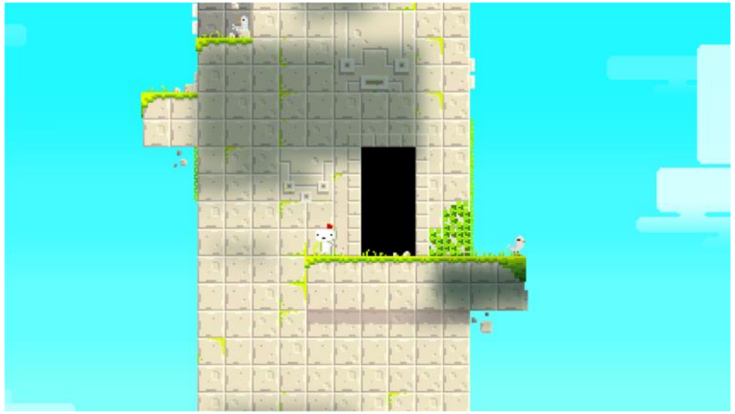
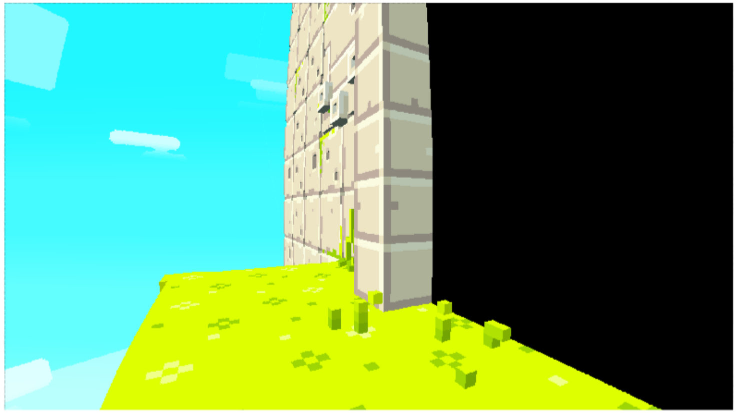
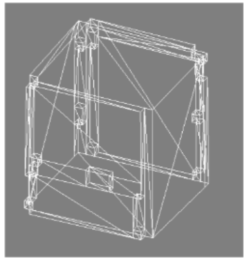
MonoGame graphics

- Allows platform-specific features
- Fine tuning for platform
- Assets optimized per platform (swizzled, tiled…)
- Shaders rewritten and optimized for each platform

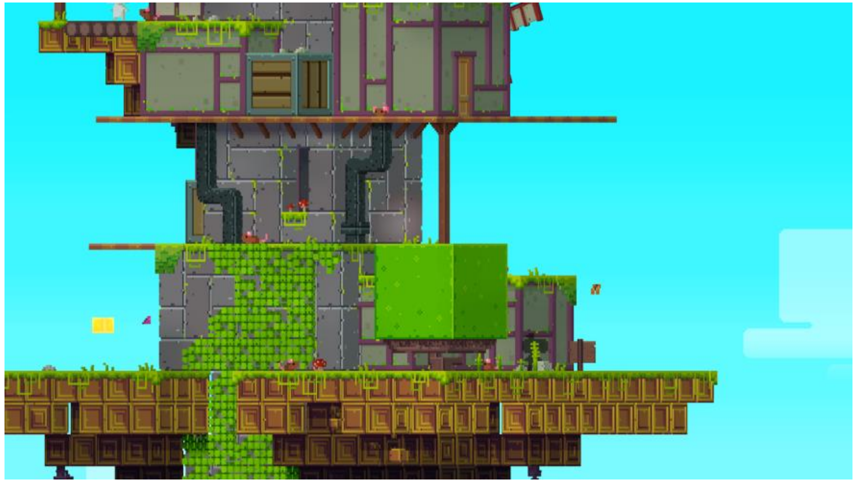
Optimization (GPU)

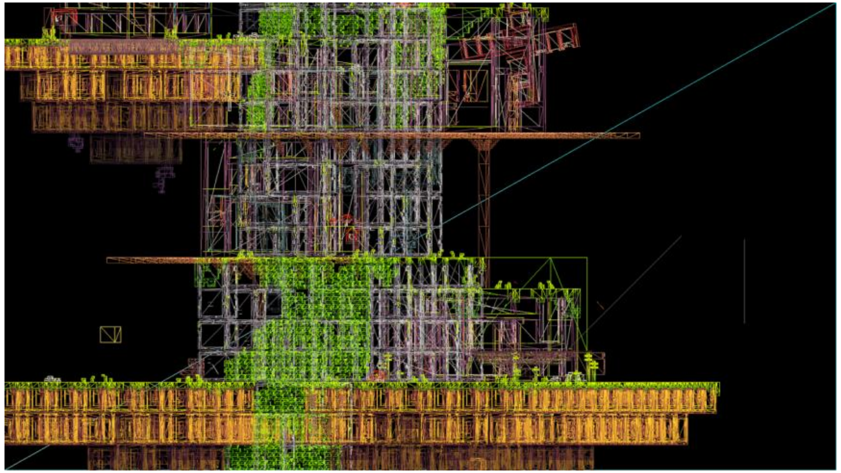
- FEZ doesn’t look like a GPU intensive game.
- That’s wrong
- “Trile” (tri-dimensional tile) made of 16x16x16 “trixels” (tri-dimensional pixel)
- Built sculpting a solid trile, removing trixels. Generates a lot of geometry
- Two pass rendering

Optimization (GPU)

Optimization (GPU)





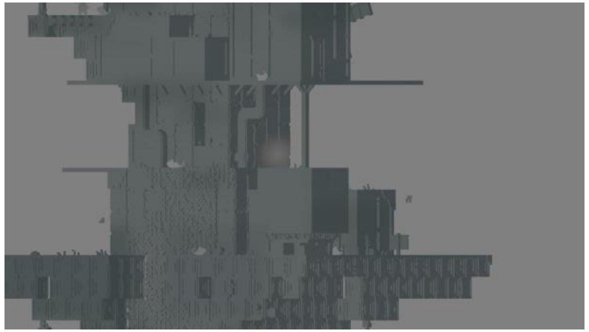
Optimization (GPU)

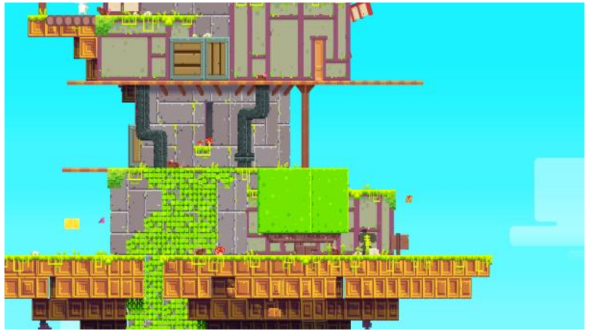
- Excessive geometry was choking vertex shaders
- Only one (while moving) or two (while rotating) trile faces visible most of the time
- Quick rejection of non-visible faces in CPU (~⅚ of the total geometry)
- Proper instancing
- Huge speedup

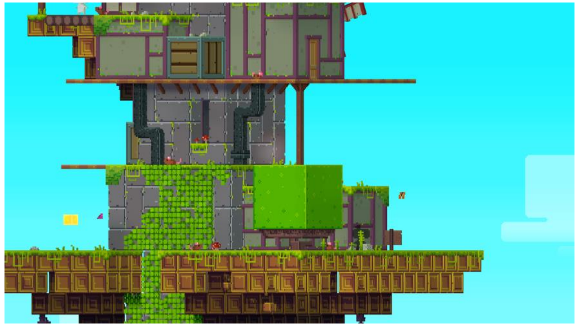
Optimization (GPU)

- Complex effects were choking the pixel shaders
- Simplified code paths
- Wrote specific “fast path” shaders to avoid branching
- Moved calculations to vertex shader when possible
- Most shaders unnecessarily used “discard”

Optimization (GPU)







New features

- Cross-save (additional slot)
  - Transparent for the user if online
  - No progress merge, just latest data
  - Allows play offline, syncs when online
- Stereoscopic 3D
  - Based on existing red/blue mode
  - Required extra tweaks

Questions?