GDC 12

Cubes All The Way Down

Renaud Bédard
Programmer, Polytron

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

About Me

- Started 3D programming in VB6 & Truevision3D ~ 2001
- Started working with Phil Fish on FEZ in April 2007
- FEZ gets 2 noms and 1 win (Visual Arts) at IGF'08
- Bacc. in Computer Science at UQÀM in late 2008
- Worked full-time since then at Polytron
  - FEZ is first commercial title and full-time “industry” “job”

Hi! Thanks.
Tech post-mortem for FEZ (XBLA, coming out shortly)

Me :

Always been programming, elementary school.

Started 3D high school out of interest, game makin’.

Mostly toyed with visual effects, shaders.

Met Phil on Internet, needed programmer for upcoming project,

Wanted to make 1st game around that time, interested in his visual style/design

3-month prototype gets IGF attention

Finish Bacc. Comp sci

Full-time for 4 years

About FEZ

- 2D/3D Exploration Puzzle Platformer (“mystroidvania”)
- Pixel art where the pixels are 3D *trixels*
- Platforming in 2D, but across all 4 orthographic views
- Built in XNA/C# from the start
  - Spanned 5 XNA versions, from 1.0 to 3.1!

If you’re here, probably heard about it...
Lucky that nearly everyone did!

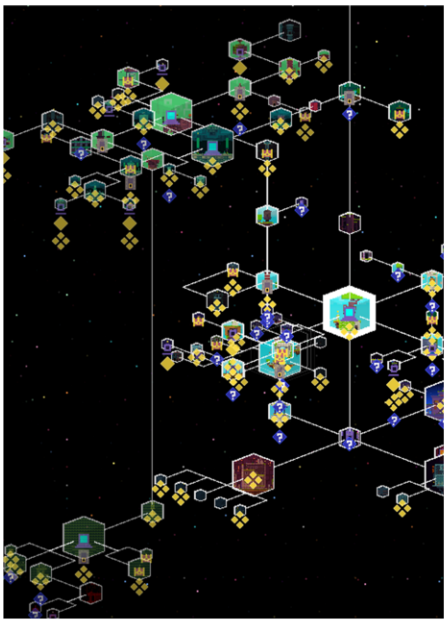
Game Footage

To give you an idea of the world aesthetic in action…
Platforming is pretty "realistic" in that it's not very floaty
Gomez (white guy) is not a ninja
Goal is to collect golden cubes to rebuild an ancient artifact
Interactive levels, navigational puzzles

GAME DEVELOPERS CONFERENCE® 2012 MARCH 5-9, 2012 WWW.GDCONF.COM

World Structure

- Big branching mess, 157 areas total
- Each is an isolated level
  - But level transitions are made “seamless”



World is network of interconnected, but self-contained levels

150+ little rooms and larger areas

To give sense of place, level transitions “in the sky” are “seamless”
Level loading & setup happens during transition, in another thread

You can see where you went in map, nodes unlock as you go

Map is pretty dense, rendered isometric, and 4-sided nature of game -> need help from game to navigate

How are levels "cut up"?

- Art- and level-production wise, tiles are easier
- Plain old 2D tilemap doesn't work : we need 3D tiles
  - …triles?
- 16x16x16 voxels (*trixels*) chosen as an arbitrary fixed size

World built out of repeated cubic chunks

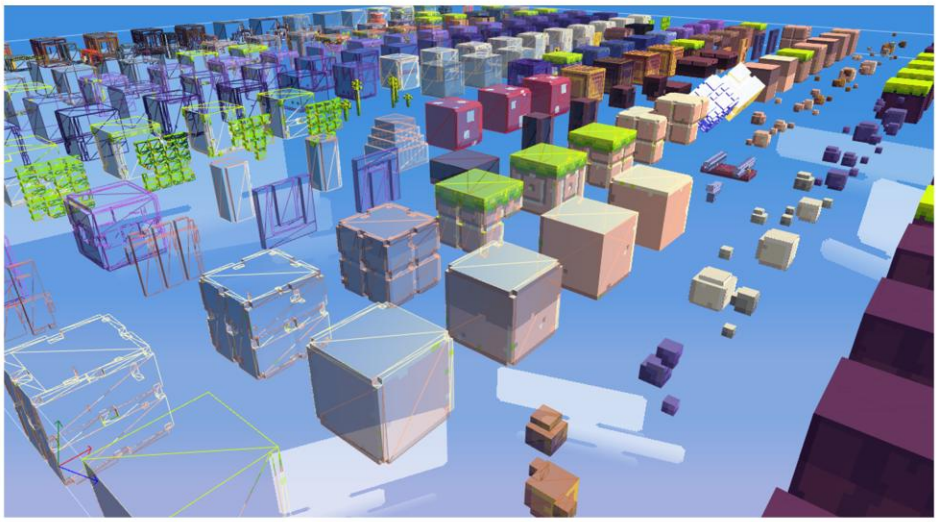
- Pixel art aesthetic made it natural to consider a tile-based world (easier to build)

- Megatextures + unique geometry : too ambitious for 1 artist and 1 coder, non-reusable assets = scary

- Axis-aligned tiles also = easier culling (possibly even hierarchical)

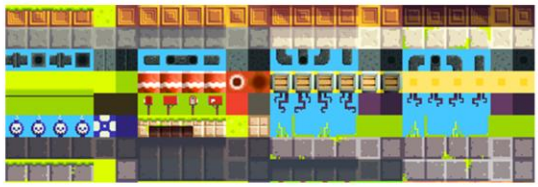
- So how are they organized?

"Nature" Trile Set (302 triles)



- "Trile Sets" = all the possible "triles" that can appear in a single level.
- 17 of them in the game
- I'll go over how these are modeled and textured.

Texturing Triles



- Per-trixel coloring?
  - Implies good integrated painting tools
- Cubemaps! (*freebloods*) F R B L U D
  - Triles need to be convex
  - Cubemaps are 16x16x6 32bpp = 6kb uncompressed!

- No palette limitation by design
- Phil is used to Photoshop

- Emulating color manipulation tools (gradients, color spaces, palettes, layers) is hard/time-consuming

- CONVEX : Pixels copied to inside faces for concave, but it's usually impossible to see in isometric views
    - Also : Small triles, usually convex

- One discrete texture per trile = lots of texture switching
    -> Atlasing is a must (as shown)

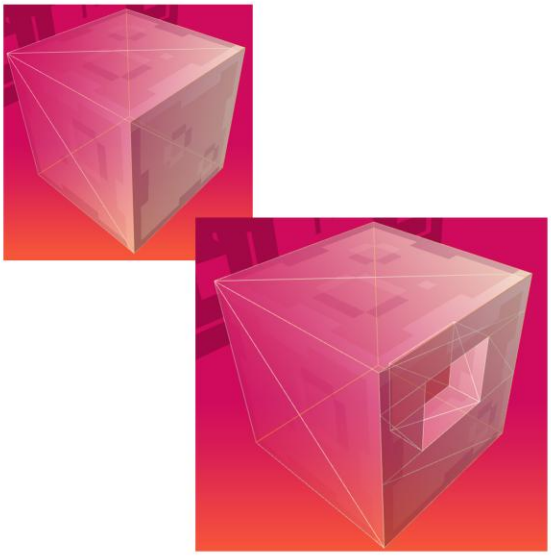
Modelling Trile

- First trile concepts made in Google Sketchup (2007)

Sketchup :

- Good to test out visual styles initially
- Not made to carve this kind of geometry
    - Took Phil forever to do it
- Can't export with the free version

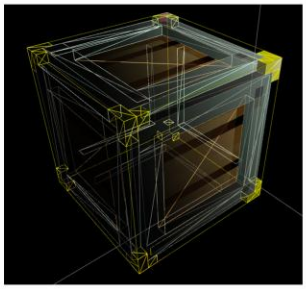
Trixel Sculpting

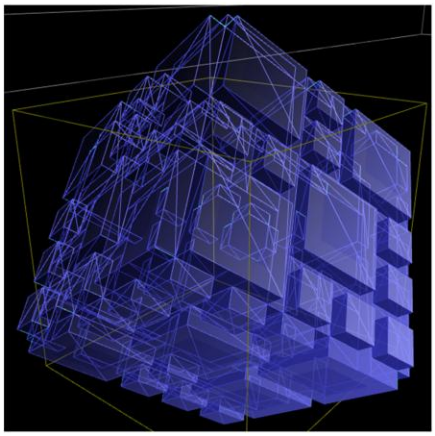


- Trixel sculpting in the editor
- Realtime mesh simplification
- Start with 8-face cube and carve chunks out
- Texturing is managed externally

Blocky But Detailed

- Average of 136 polygons per trile
- Typical scene : 50,000 polygons
  - But anywhere from ~5000 to ~250,000

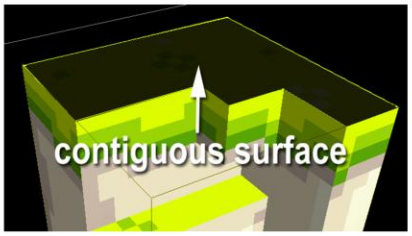


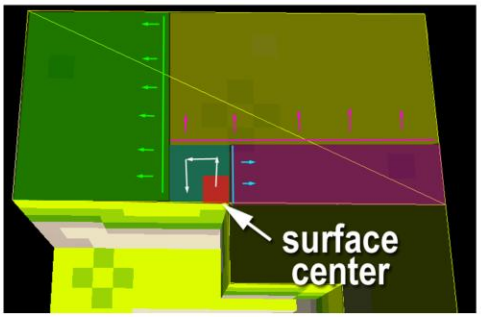


All surfaces are planar, but
there can be pretty complex triles, and
it shows when the world rotates.

Mesh Simplification

- Extrapolate contiguous surfaces from trixels
- For each surface,
  - Enumerate biggest rectangles in that surface
  - Each rectangle becomes a plane





Trile made of trixels

I keep track of presence like voxels (there or not)

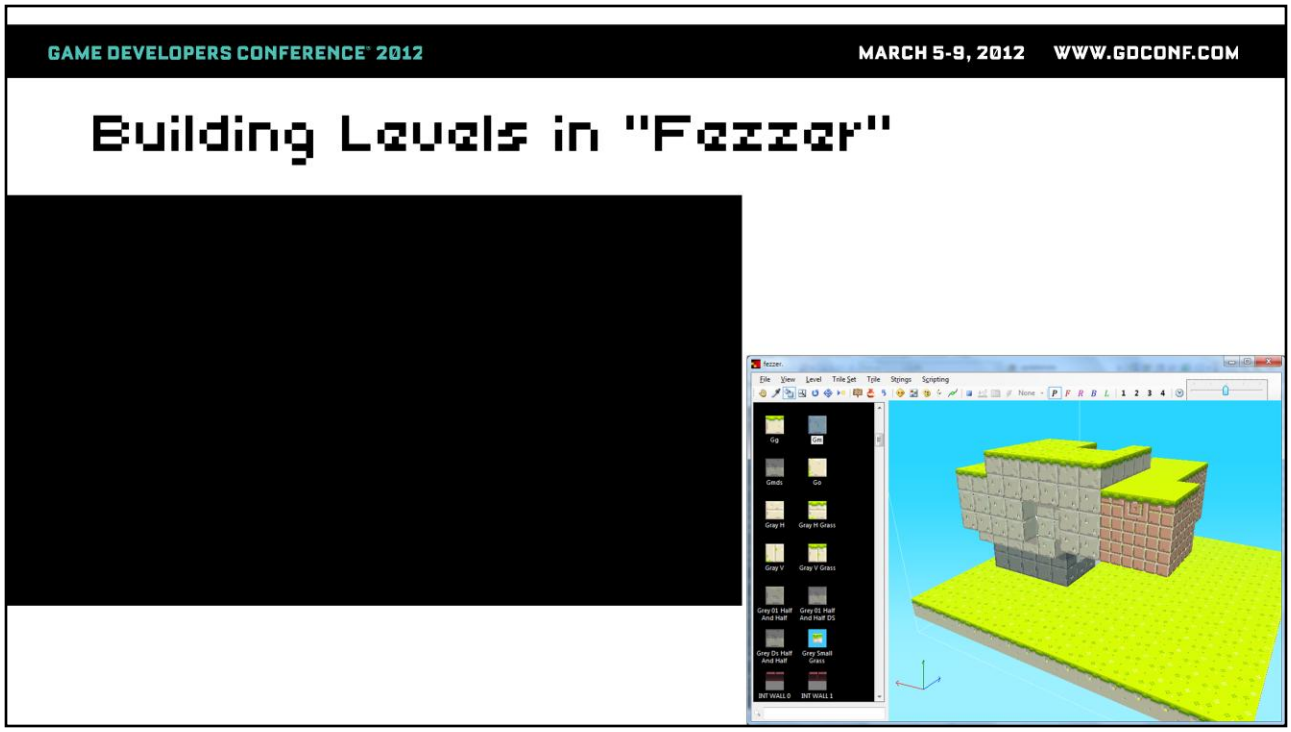
From that,

- Detect all the outwards-facing faces for trixels that have no neighbour in that direction,

- List contiguous surfaces

From surfaces,

Use 2D spiral traversal from the center of the surface to tag the biggest possible rectangle in each surface.



Much like sculpting triles, but for the whole level!

All the blocks = white & untextured

Can build the level like Legos

Then load a trile set and “paint” it.

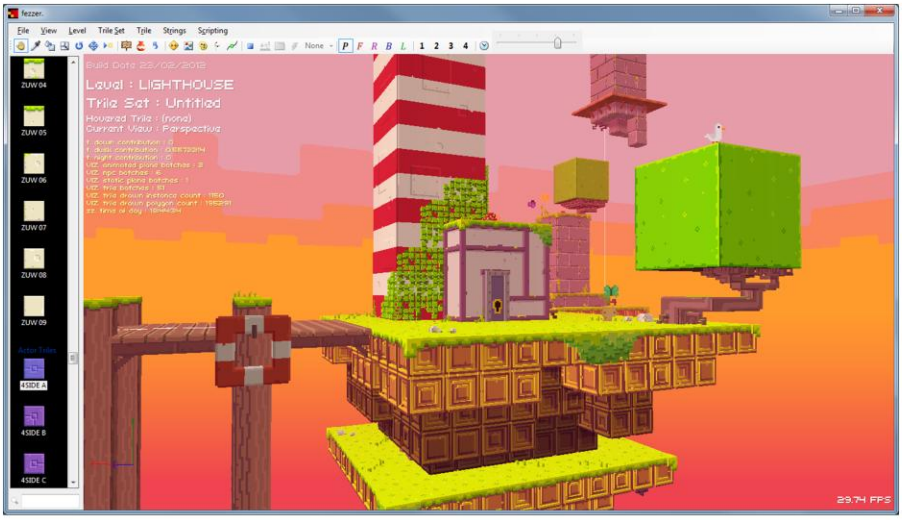
Also (tools) :

- Group stuff together
- Move it around
- Rotate triles, etc.

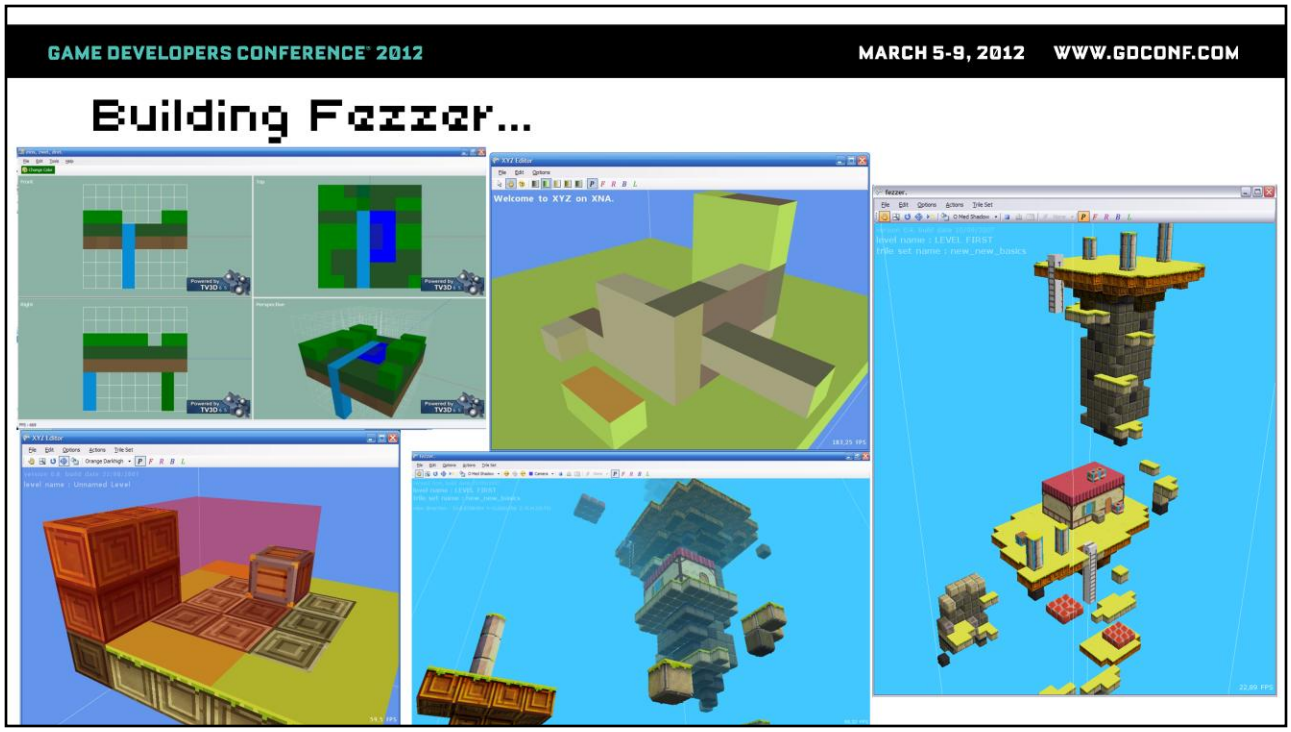
Ability to see level from four sides at all times to see how it plays.

Everything’s done from the editor.

Completed level in final Fezzer



I'm not an artist nor a level designer
Here's what Phil can do with the tool
(and his mad pixel art skillz)



A couple shots “under construction” shots of Fezzer

First prototype not in XNA, in TV3D because it’s what I knew (yay watermarks)

More of a CAD interface, with 4 views, and per-trile coloring

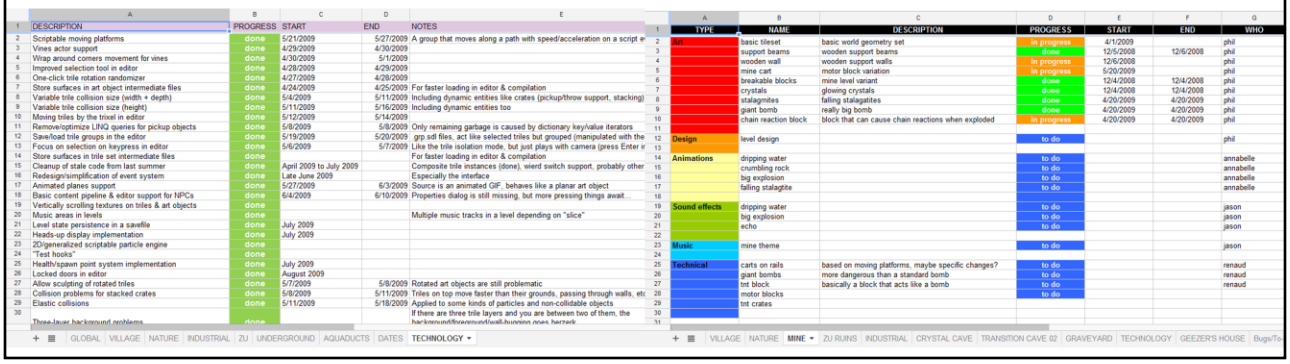
Phil preferred face extrusion

Second prototype fullscreen, face selection & extrusion; but no triles yet; just faces & colors

Triles came in after, with cubemap textures
Then trixels! And this is the IGF’08 level.

Feature tracking

- Google Docs all the way!
- Sprints planned 1 week to 1 month ahead
  - Do basic feature first, revisit & polish later



DESCRIPTION	PROGRESS	START	END	NOTES
Scriptable moving platforms	done	5/21/2009	5/27/2009	A group that moves along a path with speed/acceleration on a script e
Vines actor support	done	4/29/2009	4/30/2009	
Wrap around corners movement for vines	done	4/30/2009	5/1/2009	
Improved selection tool in editor	done	4/28/2009	4/29/2009	
One-click trile rotation randomizer	done	4/27/2009	4/28/2009	
Store surfaces in art object intermediate files	done	4/24/2009	4/25/2009	For faster loading in editor & compilation
Variable trile collision size (width + depth)	done	5/4/2009	5/11/2009	Including dynamic entities like crates (pickup/throw support, stacking)
Variable trile collision size (height)	done	5/11/2009	5/16/2009	Including dynamic entities too
Moving triles by the trixel in editor	done	5/12/2009	5/14/2009	
Remove/optimize LINQ queries for pickup objects	done	5/8/2009	5/8/2009	Only remaining garbage is caused by dictionary key/value iterators
Save/load trile groups in the editor	done	5/19/2009	5/20/2009	grp xml files, act like selected triles but grouped (manipulated with the
Focus on selection on keypress in editor	done	5/6/2009	5/7/2009	Like the trile isolation mode, but just plays with camera (press Enter ir
Store surfaces in trile set intermediate files	done			For faster loading in editor & compilation
Cleanup of stale code from last summer	done	April 2009 to July 2009		Composite trile instances (done), wierd switch support, probably other
Redesign/simplification of event system	done	Late June 2009		Especially the interface
Animated planes support	done	5/27/2009	6/3/2009	Source is an animated GIF, behaves like a planar art object
Basic content pipeline & editor support for NPCs	done	6/4/2009	6/10/2009	Properties dialog is still missing, but more pressing things await...
Vertically scrolling textures on triles & art objects	done			
Music areas in levels	done			Multiple music tracks in a level depending on "slice"
Level state persistence in a savefile	done	July 2009		
Heads-up display implementation	done	July 2009		
2D/generalized scriptable particle engine	done			
"Test hooks"	done			
Health/spawn point system implementation	done	July 2009		
Locked doors in editor	done	August 2009		
Allow sculpting of rotated triles	done	5/7/2009	5/8/2009	Rotated art objects are still problematic
Collision problems for stacked crates	done	5/8/2009	5/11/2009	Triles on top move faster than their grounds, passing through walls, etc
Elastic collisions	done	5/11/2009	5/18/2009	Applied to some kinds of particles and non-collidable objects
				If there are three trile layers and you are between two of them, the
Three-layer background problems	done			background/foreground/wall-hugging goes berzerk

GLOBAL | VILLAGE | NATURE | INDUSTRIAL | ZU | UNDERGROUND | AQUADUCTS | DATES | TECHNOLOGY

TYPE	NAME	DESCRIPTION	PROGRESS	START	END	WHO
Art	basic tileset	basic world geometry set	in progress	4/1/2009		phil
	support beams	wooden support beams	done	12/5/2008	12/6/2008	phil
	wooden wall	wooden support walls	in progress	12/6/2008		phil
	mine cart	motor block variation	in progress	5/20/2009		phil
	breakable blocks	mine level variant	done	12/4/2008	12/4/2008	phil
	crystals	glowing crystals	done	12/4/2008	12/4/2008	phil
	stalagmites	falling stalagatites	done	4/20/2009	4/20/2009	phil
	giant bomb	really big bomb	done	4/20/2009	4/20/2009	phil
	chain reaction block	block that can cause chain reactions when exploded	in progress	4/20/2009	4/20/2009	phil
Design	level design		to do			phil
Animations	dripping water		to do			annabelle
	crumbling rock		to do			annabelle
	big explosion		to do			annabelle
	falling stalagtite		to do			annabelle
Sound effects	dripping water		to do			jason
	big explosion		to do			jason
	echo		to do			jason
Music	mine theme		to do			jason
Technical	carts on rails	based on moving platforms, maybe specific changes?	to do			renaud
	giant bombs	more dangerous than a standard bomb	to do			renaud
	tnt block	basically a block that acts like a bomb	to do			renaud
	motor blocks		to do			
	tnt crates					

VILLAGE | NATURE | MINE | ZU RUINS | INDUSTRIAL | CRYSTAL CAVE | TRANSITION CAVE 02 | GRAVEYARD | TECHNOLOGY | GEEZER'S HOUSE | Bugs/To

After basic interface implemented, features came in organically.

Either face to face discussion or email, when a sprint is started.

Phil reports bugs, we prioritize and fix. Can fix on the spot if simple enough.

Features are pushed or dropped based on importance or complexity.

Tracking as simple as you can make it and still be intelligible

Too complicated tracking means you won't use it,

Too simple means it won't help you.

Designer vs. Programmer Dynamic

- Phil dictated design, but I implemented
  - As the programmer, YOU know what's possible
    - Time-wise, performance-wise, and in regards to your abilities
- You can always say no, but it's a discussion
  - FEZ is a "game d'auteur", designer associates with it deeply
  - It's all about mutual respect

If phil says "it would be really cool to have animated trixels for NPCs!",
I'm the one to say "won't happen"

There's always tradeoffs.

A note on Version Control

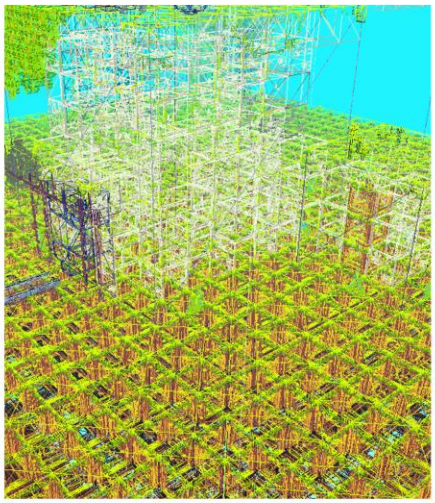
- Holy shit you guys, use version control.
- Used local NAS with batch file madness for whole 2009
  - Lost content, overwritten files, flaky backups
  - "I'm just one guy" and "I have infinite undo levels" are *not* good reasons
  - Remote SVN/Git/Hg servers are cheap, and they WILL save your ass

Thanks Nathan @ Capy for convincing us to make the jump!

How do we draw all that stuff?

- Lots of tiny blocks
- Geometry gets pretty dense
- Potentially lots of overdraw in 2D views
- Ideally 60 FPS on Xbox 360

⇒ By **culling** and **batching** efficiently



Rendering Trles : Culling

- Orthographic views, usually looking into an axis
  - Peel the frontmost layer(s)
  - At most two layers if mid-rotation
- Cull within view frustum
- When moving, only invalidate triles for the screen's moving border
  (don't recull everything unless necessary)

The world is on a grid
The grid is axis-aligned and regular
All triles are individual units
⇒ I can cull them independently

How Culling Works

- Cell content is cached
- For each screen cell
  - Start at camera depth
  - Mark trile to be drawn
  - Walk into screen
  - Stop at solid, non-seethrough & non-offset trile

When rotation occurs,

Each 2D cell in level (as seen from screen) caches contained triles

to speed up culling

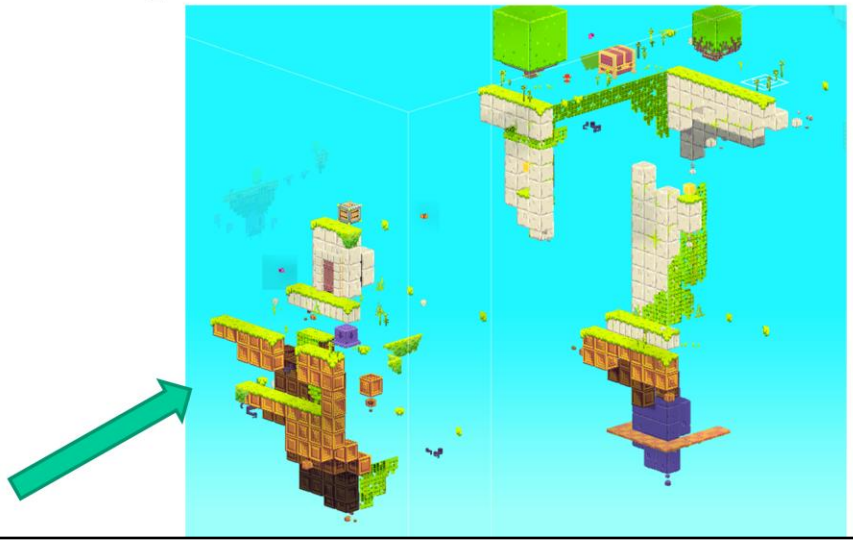
Seethrough triles : vines, doors, grass strands

Offset triles : If moved, we can see behind (like that crate)

Most triles are solid & occupy entire cell

Culling Disabled

Culling Enabled (~3X less tiles)



Absolutely no difference visually from non-culled version (in the right viewpoint)

Rendering Trilos : Batching

- Draw calls are expensive
- All triles are independent (*so indie*)
  - Can't just throw everything at the GPU

⇒ Batching is a must
  - But dynamic culling means dynamic batching...
    - Is there a simple way?

Preparing and sending draw calls is very costly,
moreso in XNA.

Can't batch everything once,
batch contents keep changing as we cull.

Rendering Triles : ~~Batching~~ Instancing

- Lots of instances of a single trile type in a level
- Few things specific to an instance
  - XYZ Position, Y rotation : 4 single-precision floats
- vfetch instancing on Xbox 360
  ≈ Shader Model 3.0 Hardware Instancing on PC
- Max. 256 instances of the same trile per draw call

Instancing to the rescue!

Instancing is :
Taking a redundant model,
Adding a set of small instance data,
And throw them as a batch to the GPU.

226 => Theoretical maximum of 256, but some other VS constants taken by uniform parameters or constants

How-to : GPU Instancing (Xbox)

- A static vertex buffer
  - Vertex data of the template trile (lightweight)
- A dynamic index buffer
  - One set of indices for each instance
  - Offset indices as if you'd have as many clones in vertex buffer
- An instance data buffer
  - Actually a 4D vector or 4x4 matrix array in vertex shader constants
  - Contains the instance-specific data

Three things needed for geometry instancing.

VB is light despite a lot of geometry output!
Index buffer is larger, can't avoid it, but still only integers.
At most 16 floats (4x4 matrix) of data per instance.

Xbox Instancing Vertex Shader

```
int vertexIndex = mod(IN.Index, VertexCount);            // Index is an automagic vertex input semantic
int instanceIndex = IN.Index / VertexCount;              // VertexCount is uniform parameter fed by app
asm
{
       vfetch position,          vertexIndex, position0
       vfetch normal,            vertexIndex, normal0
       vfetch textureCoordinate, vertexIndex, texcoord0
};                                                       // vfetch magic gets appropriate vertex data

float4 InstanceData = InstanceDataArray[instanceIndex];

float sinPhi, cosPhi;
sincos(InstanceData.w, sinPhi, cosPhi);                  // W component contains Y-axis rotation angle
float4x4 instanceMatrix =                                // Recompute instance transformation matrix
{
              cosPhi,       0,            -sinPhi,      0,
              0,            1,            0,            0,
              sinPhi,       0,            cosPhi,       0,
              InstanceData.xyz,                         1
};
```

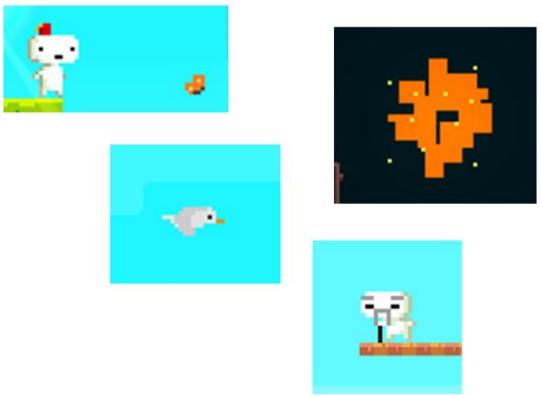
Sorry if this slide is dense/technical.

Our approach :

- 4D vector per instance
- W component holds Y-axis rotation
- Instead of whole 4x4 transformation as parameter, rebuild it inside VS
- 4x more instances per batch!

Other Stuff : Planes/Decals

- Where sclupting doesn't matter, or sprite animations



Gomez's house + many other house interiors = mostly planes

- Can't tell the difference
- Easier to draw whole slab of 2D art than split it in 16x16 chunks

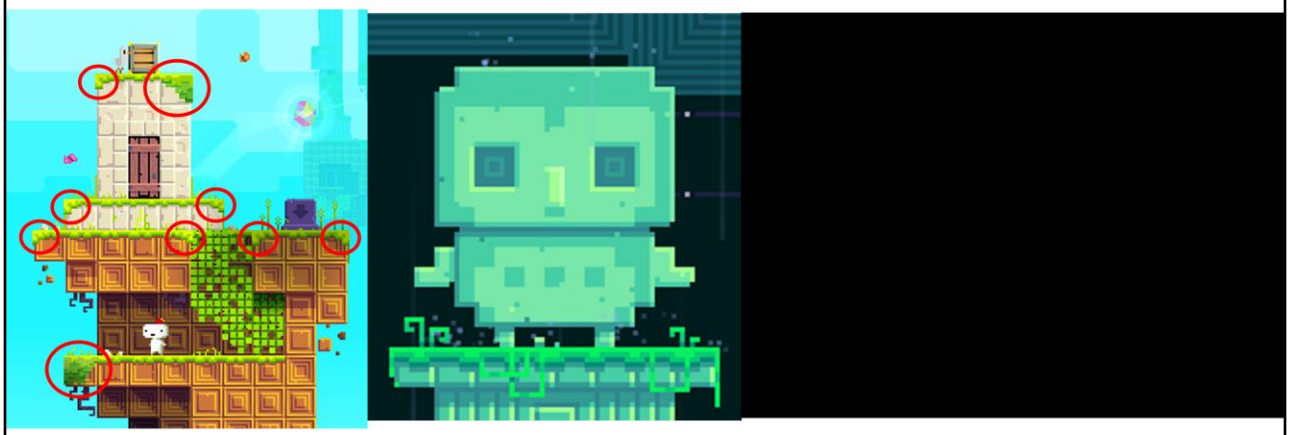
Other planes :

- Explosions
- NPCs & wildlife
- Gomez

No support for animated trixels. (shout out to Voxatron)

Other Stuff : Art Objects

- For small overlapping details, or unique bigger landmarks

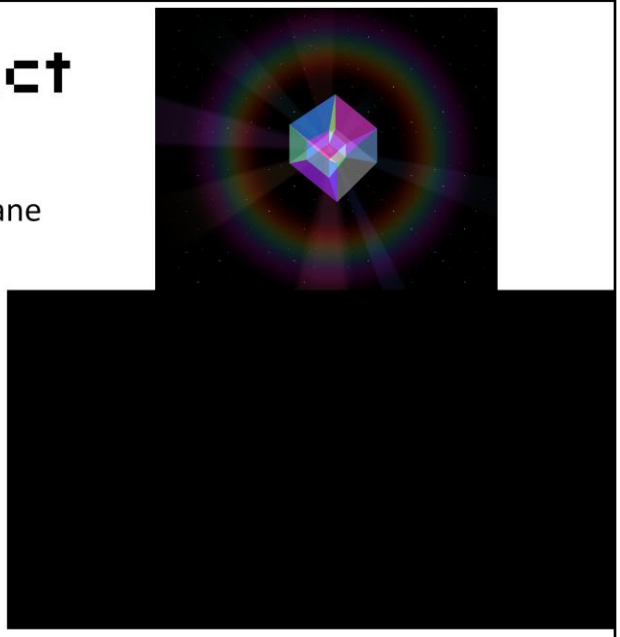


- Sculpted & textured like triles, but bigger
- When splitting in 16x16 chunks is impractical
- Also instanced (lots of duplicates)

All circled little green thingys are art object
- Don’t fit in grid model, overlap triles

DOT the Tesseract

- 4D hypercube fairy
- Continually rotates about the X-W plane
  - Done in C# (on the CPU)
  - 4x4 matrix works for rotation
- Faux 4D to 3D projection
  - Further out in W axis = smaller in 3D
- Rendered in orthographic projection like everything else
- 96 vertices, 144 triangles
  (no intersection in 4D space)



You might've noticed DOT in earlier shots/videos

She's FEZ's "Navi" or "Fi"

Story-wise it makes sense that she's one-step-beyond what Gomez can see, hence 4D

Collision Management

- Triles *are* the collision map
  - Each trile has a type
    - Can be per-face too!
- Four types
  - Immaterial (blades of grass)
  - No collision (background elements)
  - Top-collide (most platforms)
  - All-collide (blocking boundaries; rare)

World mostly made of triles
=> Why not using them directly for collision info?

No-collide => Unlike immaterial, push Gomez forwards
All-collide => interior closed-up areas.

Invisible Collision Trles

- Art objects need to collide too
  - But they're not made of triles!
- Filled with invisible triles
  - No collision, or
  - Top-only collision

These are costless to the renderer.
Only used for collision.

In that case,
Triles really act as a collision map...

Collision Lookup

- Similar as the culling process
  - Collide with what you see!
  - Peel back from view
  - Keep peeling back if hitting
    - Immaterial triles (i.e. grass strands)
    - Non-grid-aligned triles

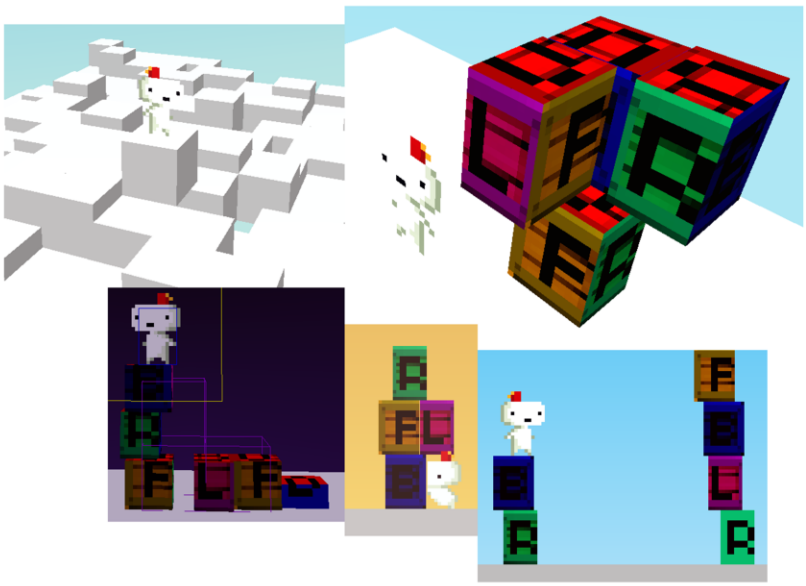
Otherwise, point-to-line with first solid trile edge found (three points), one axis at a time.

Non-Grid-Aligned?

- Triles can be moved arbitrarily
  - Stored in cell that contains its center
- Collision tests look up neighbours
  - Only one, away from cell's center
  - Only if no hit in current cell
  - Once found, point-to-line
    (using appropriate size & offset of the trile)

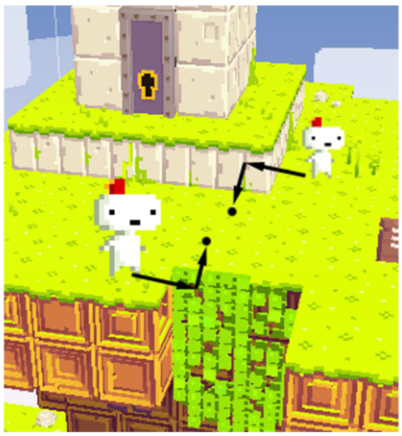
Tests...

- Scaling can also be arbitrary
- As long as triles are no smaller than half of Gomez's size
  (limitation of using point collision)



How Gomez moves around

- Movement is along on the view plane
- Depth correction rules
  - Gomez should stay visible, always
  - Gomez should never walk in mid-air
- Otherwise, don't touch his depth



- During view rotations, movement & time are suspended

Background Mode

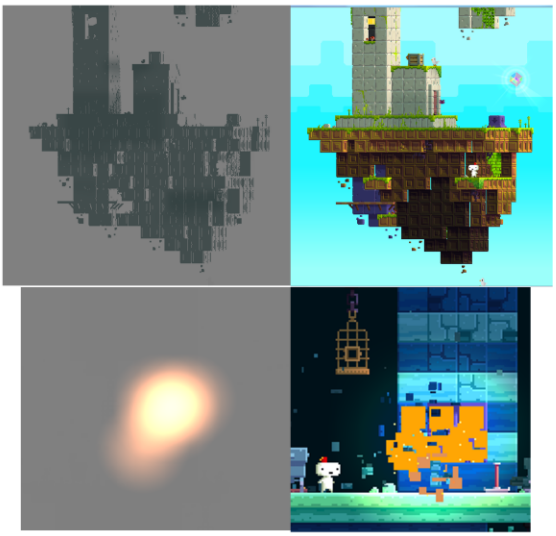
- If Gomez is behind the level post-rotation
  - IGF'08 build : Panic & QTE!
    - ⇒ Stressful, kills the mood, generally dumb

- Final build
  - Silhouette rendering
  - Low-pass music
  - Limited actions

Idea is to
communicate smoothly that he shouldn't be there.

Lighting Pre-Pass

- Per-face direct/diffuse light
- Ambient light = sky background color
  - Cloud shadows end up blueish
- Shadows and additional lights added
  (in screen-space)
- All done in a lighting pre-pass
- Blended in Modulate2X mode
  - so that it can light up and shadow



Lighting is done in 2D with planes,
NOT point lights

Ended up doing a pre-pass :
- Shadows can max-out
- Lights can cancel shadows without altering colour
- Lights can add, and be coloured

Without pre-pass, albedo information is lost in shadows
Con : Need to draw everything 2X

This is 8x speed day/night cycle

Time is mostly visual/aural/atmospheric,
but has gameplay implications in some areas

Sky color defines :
- Sky layer tinting color (“fog color”)
- Ambient light color

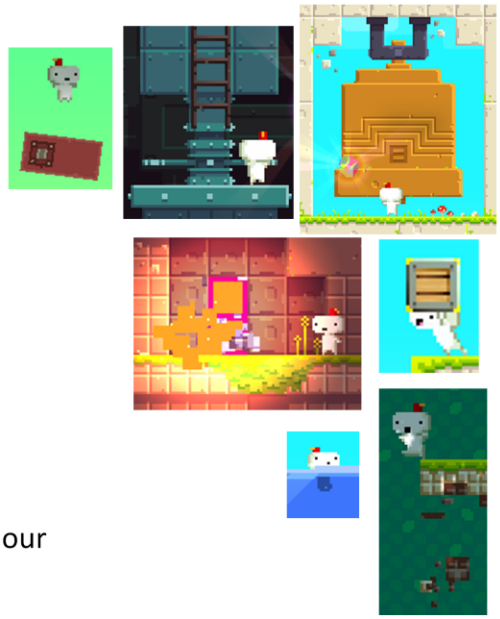
Clouds & cloud shadows become less apparent at night

Additional post-process during dusk/dawn/night
...to make colors pop and dull colors at night

Talk about texture sampling!

World Interactions

- Gomez can :
  - Grab/push/pull objects
  - Rotate parts of the world independently
  - Make blocks crumble under his weight
  - Grab ledges all around platforms
  - Interact with one-off puzzle objects
  - Swim in water & drown in toxic liquids
  - AND MUCH MUCH MORE
- 56 different “action classes” control his behaviour



Every action type has associated animation too.

Action "Classes For States"

- All player action classes derive from a base abstract class
- Not all mutually exclusive
  - "Fall" is an action that is evaluated as long as there's gravity
- They know how to chain from state to state

```
protected virtual void TestConditions()
{
}

protected virtual void Begin()
{
}

protected virtual void End()
{
}

protected virtual bool Act(TimeSpan elapsed)
{
    return false;
}
```

Actions classes know when they need to start taking control

Basically avoids doing a bit "switch" or having a million booleans.

Example : WalkRun.cs

```
protected override void TestConditions()
{
    switch (PlayerManager.Action)
    {
            case ActionType.Sliding:
            case ActionType.Idle:
            case ActionType.Teetering:
            case ActionType.Grabbing:
            case ActionType.Pushing:
            case ActionType.LookingAround:
                // If grounded and pressed movement keys
                if (PlayerManager.Grounded && InputManager.Movement.X != 0 &&
                    PlayerManager.PushedInstance == null)
                {
                        PlayerManager.Action = ActionType.Walking;
                }
                break;
    }
}
```

They are backed by an enumeration that has all the different states.

This is how they find out when to start acting,

Example : WalkRun.cs

```
protected override void IsActionAllowed(ActionType type)
{
    return type == ActionType.Running || type == ActionType.Walking;
}

protected override bool Act(TimeSpan elapsed)
{
    // Transform input to physics impulses in a helper class
    MovementHelper.Update((float)elapsed.TotalSeconds);

    if (MovementHelper.Running)
           PlayerManager.Action = ActionType.Running;
    else
           PlayerManager.Action = ActionType.Walking;
}
```

...whether or not the action class is active,

And what's going on when they act.

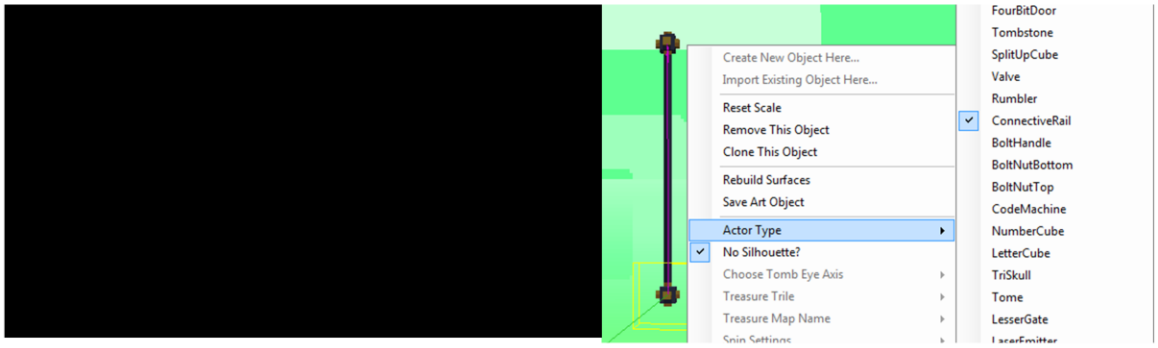
Additional Flags for Actions

```
public static class ActionTypeExtensions
{
    public static bool IsAnimationLooping(this ActionType type)
    {
        switch (action) { /* ... */ }
    }

    public static bool DisallowsRespawn(this ActionType type)
    {
        switch (action) { /* ... */ }
    }

    public static bool PreventsRotation(this ActionType type)
    {
        switch (action) { /* ... */ }
    }
    /* ... */
}
```

There's also many other flags based on the enums that defines other behavioral characteristics.

Actors : Dynamic World Entities

- Spinning blocks, moving platforms, ladders, interactive structures, etc.
- Hardcoded behaviours with flags or parameters set in the editor
- Tradeoff for not having proper scripting support



Actors are
- tagged trile groups,
- art objects or
- even volumes or planes

that do specific things in-game.

Usually have a “manager” class that handles all instances of them in a level.

Scripting System

- Designer-friendly UI (no code!)
- Event-based

Scripts Browser

Id	Trigger	Condition	Action
10	Level.Start		Dot.SpiralAround(True, False), Dot.Say(DOT_TRIAL_A, True, False), Dot.Say(DOT_TRIAL_B, T
11	Volume[6].Enter	Game.GetLevelState != JUMP, Game.GetLevelState != LOOK, ...	Game.ShowScroll(TUTO_VINE, 5, True, True)
4	Volume[3].Enter		Level.ChangeLevelToVolume(MEMORY_CORE, 0, True, True)
6	BitDoor[38].Open		Volume[3].SetEnabled(True, False)
7	Level.Start		Camera.SetPixelsPerTrixel(3), Volume[3].SetEnabled(False, False)
8	Level.Start	Level.FirstVisit = False	Sound.ChangeMusic()
9	Level.Start	Level.FirstVisit = True	Sound.ChangeMusic(Clear Skies)
13	Gomez.Jumped	Game.GetLevelState = JUMP	Game.CloseScroll(TUTO_JUMP), Game.ShowScroll(TUTO_LOOKAROUND, 5, True, False), Ga
14	Gomez.LookedAround	Game.GetLevelState = LOOK	Game.Wait(1), Game.CloseScroll(TUTO_LOOKAROUND), Game.ShowScroll(ROTATE_INSTRU
15	Camera.Rotated	Game.GetLevelState = ROTATE	Game.Wait(1), Game.CloseScroll(ROTATE_INSTRUCTIONS), Game.SetLevelState()
23	Volume[7].Enter		Game.ShowScroll(TUTO_GRAB_LEDGE, 0, True, True)
20	Volume[3].Enter		Game.ShowScroll(TUTO_ENTER_DOOR, 0, True, True)
21	Gomez.EnteredDoor		Game.CloseScroll(TUTO_ENTER_DOOR)
22	Gomez.ClimbedVine		Game.Wait(1), Game.CloseScroll(TUTO_VINE)

Was made such that Phil can add and edit scripts without fearing syntax errors.

Here's an example of complex scripts

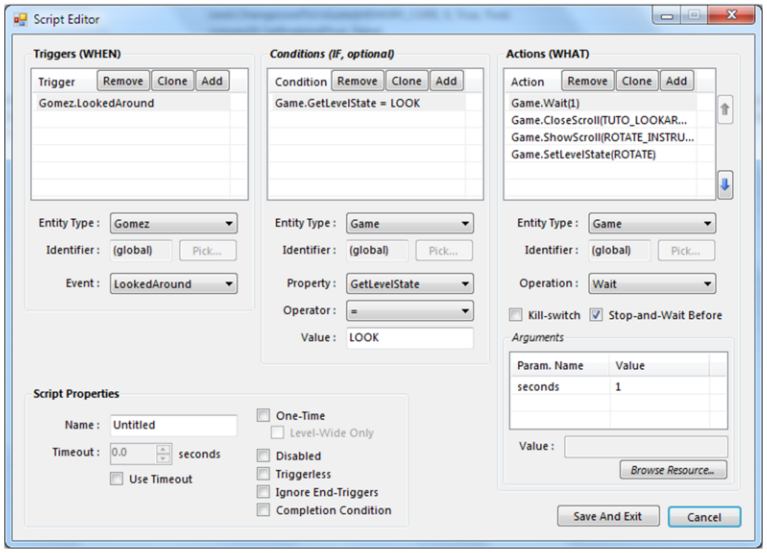
Act on Gomez actions, entering volumes of space in level, level events, etc.

Control camera, control tutorial prompts, dialogue...

Script Editor

- Triggers
- Conditions
- Actions

- And lots of WinForms controls

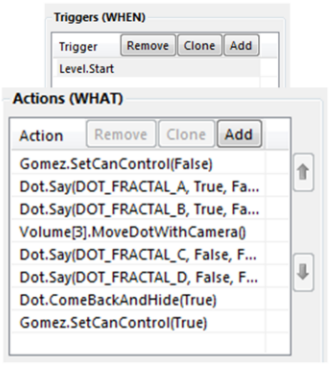


Script Example : DOT interaction

- Warn the player of a particular mechanic in a level using DOT

When Level Starts, (`Start` event on `Level` static entity)
(blocking actions executed in sequence)

- Remove player controllability
- DOT says a couple lines
- Move the camera to Volume #3 (point of interest)
- DOT says more stuff
- DOT comes back and hide
- Player regains control



This script happens once in the game :

Scripting Interfaces

```
[Entity(Static = true)]
public interface ILevelService : IScriptingBase
{
        // Events, for triggers
        [Description("When the level starts")]
        event Action Start;
        void OnStart(); // Called by the engine to trigger the event

        // Properties, for conditions
        bool FirstVisit { get; }

        // Operations, for actions (can be running over time à la coroutine)
        [Description("Smoothly changes to a new water height")]
        LongRunningAction SetWaterHeight(float height);
}
```

Level Format

- At design-time, serialized from objects to SDL
  - Similar to YAML or JSON, but better integrated with .NET
  - Tweakable by hand
  - Error resilient
    - Ignores unknown elements
    - Elements can be marked as optional
- At compile-time, binary format for performance & filesize
  - No automatic serialization, reflection is too slow on Xbox

Chose not to use XML because levels (& triles) are formed of a LOT of cells

Direct XML representation was too dense, hard to read, heavy on disk, long to load

SDL Looks Like This

- Output much more concise than equivalent XML serialization
- Serialization tags in data objects

```
public string Name { get; set; }

public TrileFace StartingPosition { get; set; }

public Vector3 Size { get; set; }

[Serialization(Optional = true)]
public float BaseDiffuse { get; set; }

[Serialization(CollectionItemName = "Trile")]
public Dictionary<TrileEmplacement, TrileInstance> Triles;
```

```
level type="FezEngine.Structure.Level, FezEngine" {
    name "ARCH"
    startingPosition {
        face "Front"
        id 16 6 5
    }
    size 30F 49F 35F
    baseDiffuse 1F
    baseAmbient 0.35F
    haloFiltering true
    blinkingAlpha false
    waterHeight 11F
    skyName "WATERFRONT"
    trileSetName "Untitled"
    volumes {
        volume key=0 {
              orientations "Front"
              actorSettings {
                  farawayPlaneOffset 5F 1F
              }
              from 3F 26F 15F
              to 4F 28F 16F
        }
```

Vector3 collapsed to single lines instead of x, y, z

Especially useful for large collections of vectors

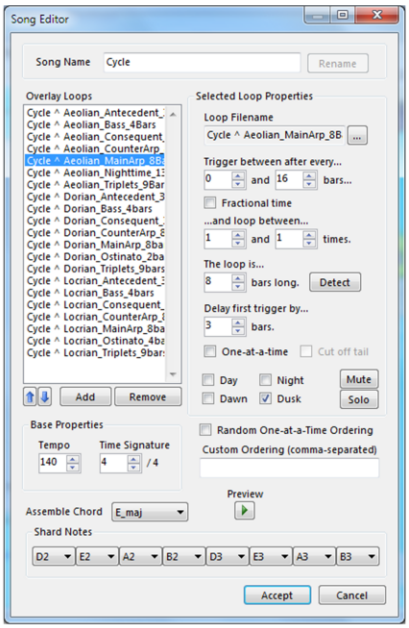
Dictionary serialization is lean if key can be collapsed to a single line

No block-end marker, easier on the eyes if you like curly braces! (and who doesn't, cave story what what)

I definitely suggest you look up more collapsed data languages like SDL and YAML.

Music System

- Written on-top of XACT
- Allows infinite, dynamic track-based songs
- Scriptable
  - Level/player events can mute/unmute tracks
- Works with time of day



Music system is a big part in FEZ, atmosphere and sense of place

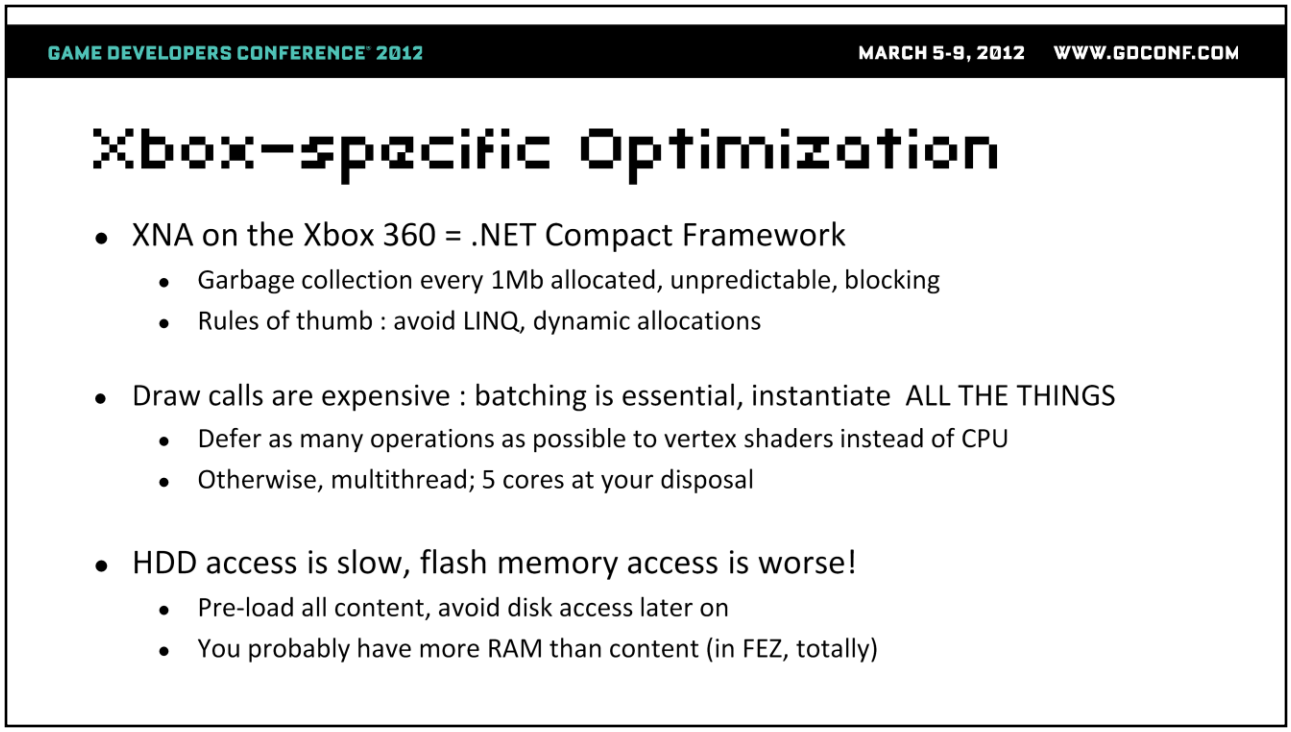
Disasterpeace used it to track all game music

Music System In Action

- "Puzzle-solving music" @ daytime

Track	Initial Delay	Duration	Inter-Play Delay
Main Arp	None	8 bars	[0, 16] bars
Counter Arp	4 bars	8 bars	[0, 16] bars
Ostinato	8 bars	4 bars	[0, 16] bars
Bass	8 bars	4 bars	[0, 24] bars
Antecedent	16 bars	3 bars	[0, 16] bars
Triplets	16 bars	9 bars	[0, 16] bars
Consequent	20 bars	3 bars	[0, 16] bars

- Night is less dense, very different sounding and still randomized :

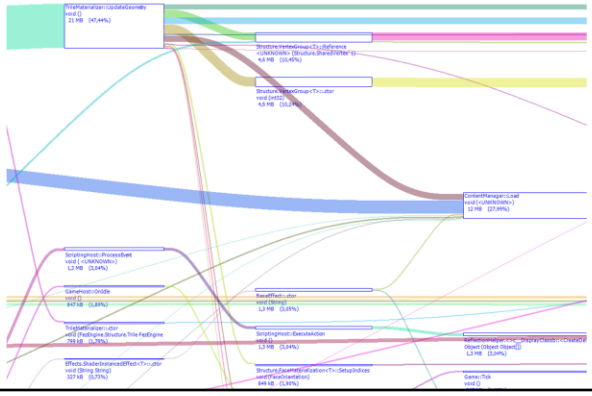


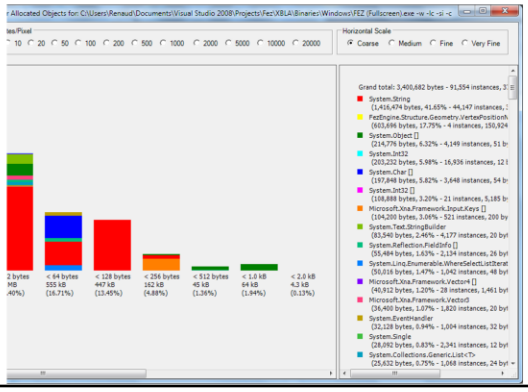
Lots of content on the Web about this

...especially since XBLIG games have the same issues

Tools : CLR Profiler

- Excellent, free memory profiler
- Allocations graphs, memory usage by class
- Good for identifying real-time garbage & why load-times stall

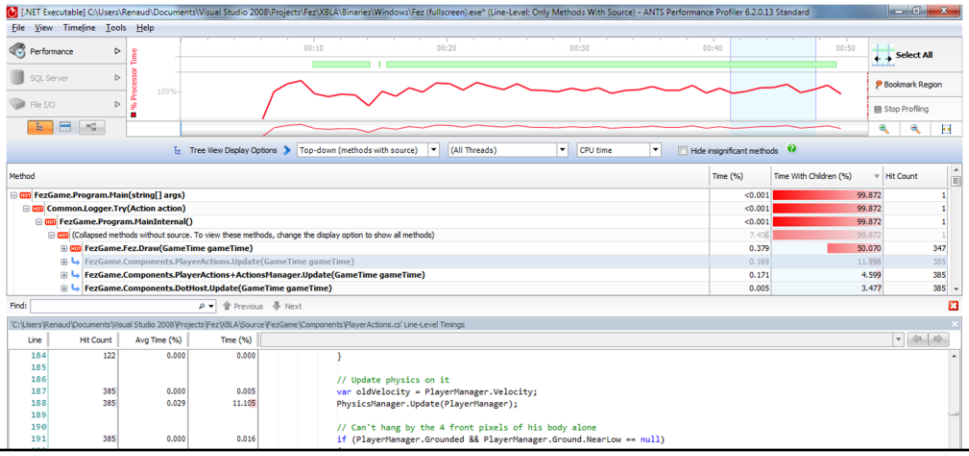




Start profiler, don't do anything, wait 20-60 seconds, check reports.
If you get allocations, that's garbage and you know where it comes from.

Tools : CPU Profiler

- Any one you want (AQTime, dotTrace...), but I like ANTS Performance Profiler a lot
- Absolutely essential for finding bottlenecks



Let the game run for a while,

Do things that you feel are expensive or slow game down,

Highlight section of the gameplay, and look at “hot methods”.

Sometimes surprising!

Explosions were slow in FEZ, I thought it was the particles

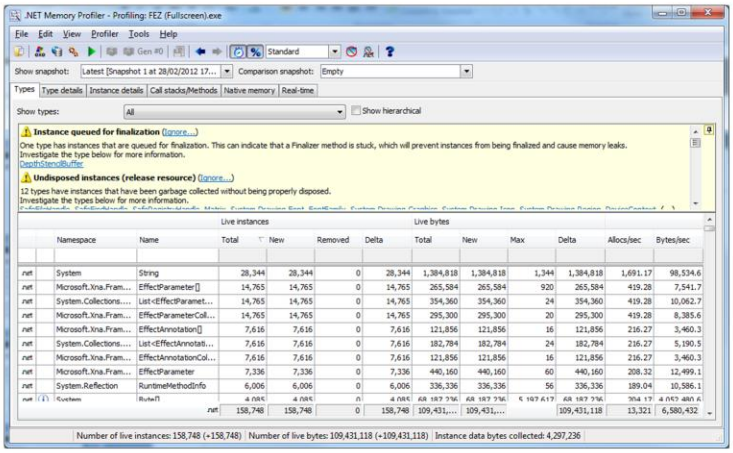
It was actually too many sounds being created.

Easy to go on optimization spree and optimize stuff that,s not worth it.

Microoptimization can be useful, just need to know where.

Tools : Analyzing Memory Profiler

- CLR Profiler is good for garbage generation, but isn't very helpful for leaks
- I used the SciTech .NET Memory Profiler
- Heap snapshot comparisons
- Insight on possible problems
- Leaky objects are identified and their creation point given



Memory leaks are possible in managed languages like C#

Usually caches that are not cleared, references lying around, delegates that are not used but kept, static stuff.

Let the game running,

Capture heap at a certain point,

Move around, change level, change context, restart game, what have you

Recapture heap

Check old references and stuff that wasn't cleared

Disposable objects not disposed,

Disposed objects not garbage collected,

Etc.

XDK Tools

All other tools worked with a PC build; what if stuff only happens on Xbox?

- xbWatson
  - Make your own measurements and output to console
- PIX
  - Frame-by-frame teardown of what's costly
  - Excellent for inspecting the draw calls & shaders
- CPU Performance Profiler in the XDK ultimately useless
  - Made for native code, not .NET games

XNA on XBLA : My Experience

- Long dev cycle meant struggle with upgrades
- No native library allowed, only .NET : can be problematic
- Some boilerplate TCR stuff handled, but still a lot to think about
- No symbols for debugging .NET assemblies in Release builds

But...

- .NET, C#, XNA and WinForms make engine & tools dev way easier
- Transition from PC to Xbox all in all fairly painless
- It's all about comfort : I couldn't have done FEZ in C++

We avoided XNA 4.0 and hit roadblocks later with known bugs

Low-quality video playback in 3.1

TCR struggles : we didn't even do multiplayer

Bugs with LIVE disconnection, wierd workarounds that everyone has to make, why?

A lesson is learned...?

- I don't think there's a way around it : a first game is HARD to finish
  - Especially if you care a lot about it
  - And let's face it, FEZ is a huge game
- Early showing was a double-edged sword
  - But later PAX/Fantastic Arcade showing were great motivators & feedback tools
- Feature creep, constant feeling that finish line is 3-6 months away
  - Making short-form "game jam" games helped learn scope control

If I had to do it again...

- Use middleware (Unity?), or hire an engine programmer
- Have real scripting support and educate artists about it
- Hot-reloadability of scripts and content edits
  - Even if C# compilation is fast, back & forth is huge waste of time
- Don't be afraid to scrap prototype code
  - 4-year-old bugs coming back to haunt you : it sucks
  - Realize you're doing a big, long project, and that it's worth the effort

Making the engine, the tools, the game at the same time is too much work

A real clean-cut engine, and whole game as scripting, robust scripting support which we didn't think we needed– needs to be done from the get go

By rebuilding you will fix those bugs without thinking about it

Have to realise that there is value to thinking it over

That's all, folks!

- Thanks for coming!
- Questions?