ScrollingBackground

christoph.wagner@live.com

Documentation as a quick overview and reference for developers. Info what this class is all about, code samples and important functions, members.

Contents

1 Abstract:

The class represents a texture showing a part of a infinite* timeline with notes and other events.Travel through time from date to date or by gestures.

1.1 Less Abstract:

The class texture shows six hours of a timeline.
The timeline repeats over and over like each day does. The limit was set to +/- 100 billion years. The date needs to be drawable on a texture, thus a limit is needed.

Four background images make a background for one day. This is called a **layer**. Seven layers are possible, each has its own configuration like color tint, show/hide, trancparency, movement

Date items(birthdays, events, notes, ...) are shown in this timeline. They are called **Notes**. They are created manually or loaded from medialibrary(special filename)

Moving in time, by touch input/one date to another/phone tile etc. lets the timeline flow to the future or past. Some layers move slower(far away) the near layers move faster, which is a nice visual effect, another layer show a scale, an overlay picture(date overview image), a overall menu for the user to disable movement...

Naming:

- class Sb: class ScrollingBackground. The main class and owner of layers, background items, date items.
- Layer: An image for a day at a depth(z-Buffer)
  Sample:

Hallo Welt

Hallo Welt

Bclogo seems to be the most versatile!

This box is made with bclogo. Corners are rounded and I love that crayon :-).

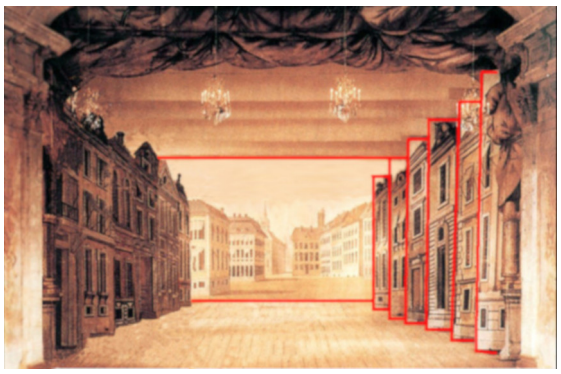
2 Moving in time

- Navigate from a note to another.
- tild the device
- gestures pan to move, hold to interact
- calender buttons
- background items like direction signs, ..

3 Layers:

A Layer is a background image for a whole day, with properties like depth, coloring, trancparency, hide/show, fade, self moving speed.

The moving speed is configureable. When the far away layers are moving slow and the near ones faster it makes a nice visual effect.



Example: Layers as stages in theatre

Layers have no click detection. But you can use background items instead. When the item is larger than about 4 hours you should create a new one.

- Sky - just sky, in the night the stars, no speed
- Clouds - far away clouds, speed is slow moving
- CloudsNear - near clouds, move little faster
- Background - main layer for background drawing, no speed
- BackgroundNear - some special background to make the moving effect look better, more speed than before
- Scale - a scale paper, plus sunrise/dawn and other dates, no speed
- Digits - the 0-24 digits of the timeline, no speed
- ScaleBelowAndDigtis - the 0-24 digits of the timeline plus a scale, no speed

Between those layers note items and events are drawn by their depth. They're between BackgroundNear and Digits.

4 About Notes:

- are faded in when loaded, fade out when unloaded
- Notes have a bucket id [bucketId0 .. bucketId47]. This bucket has a link to the note. Access the note this way, else you've to check if the note still exists.
- When the note is not visible, it can be unloaded anytime. This restriction make it possible to have nearly unlimited notes useable by the class, without loading them into memory.
- automatically load Picture Gallery formatted items like and create notes in application.
- Scaleable, Performance is no issue. Class holds about 60 notes in memory, those are about 48 possible notes for centerdate +/- 12 hours and some extra notes on the border.
  Thus even if there are thousends of notes with there texture variants no memory problem can occur.
- Create a note by Hold Gesture, then optionally use the menu to assign an image
- texture variants are greyscale, colored, hidden, transparency
- repetitions - daily, weekly, yearly, decade, century

5 Simple Item:

Import a simple texture.

5.1 Create:

```
Texture2D Tex = ..;
Point pmsTex = ..;
CommonSb.UpdateItem_Input UpdateItem_InputTex = new CommonSb.
    UpdateItem_Input();

UpdateItem_InputOwl.tex = Tex;
UpdateItem_InputOwl.centerRotation = Tex.Bounds.Center;
UpdateItem_InputOwl.depth = (int)casDepth.tex * DepthRatio;
UpdateItem_InputOwl.id = casId.tex;
// eg. daily: promille of day, 500pm -> 12pm/noon
UpdateItem_InputOwl.pm = _pmsTex.X;
UpdateItem_InputOwl.y = _pmsTex.Y;

_ScrollingBackground.Import(UpdateItem_InputTex);
```

5.2 Draw:

```
if (_ScrollingBackground.IsVisibleCalc(casId.tex))
{
        _ScrollingBackground.Draw(casId.tex);
}
```

6 Animated Item:

Show a spritesheet animation

6.1 Create

1. Create a local Spritesheet class
2. Create a local UpdateItem_Input class
3. Assign references
4. Import to ScrollingBackground

LoadContent:

```
CwaSpriteSheetSelector NowBouncing = new CwaSpriteSheetSelector(...);

CommonSb.UpdateItem_Input UpdateItemInputBouncing = new CommonSb.
    UpdateItem_Input();

UpdateItemInputBouncing.obj = NowBouncing;
UpdateItemInputBouncing.tex = NowBouncing.SpriteSheet;
UpdateItemInputBouncing.SourceRectangle = NowBouncing.SourceRectangle;
UpdateItemInputBouncing.centerRotation = NowBouncing.SourceRectangleCenter;
UpdateItemInputBouncing.depth = (int)casDepth.NowBouncing * DepthRatio;
UpdateItemInputBouncing.id = casId.NowBouncing;

UpdateItemInputBouncing.dt = DateTime.Now; //todo like TimelineNow
UpdateItemInputBouncing.y = 360;

_ScrollingBackground.Import(UpdateItemInputBouncing);
```

6.2 Update

1. Option: Check if animation is visible
2. Get a reference to the spritesheet from the ScrollingBackground
3. Update the spritesheet(source rectangle changed)
4. Assign the result to the ScrollingBackground

Update:

```
if (_ScrollingBackground.IsVisibleCalc(casId.NowBouncing))
{
                CwaSpriteSheetSelector bounce = _ScrollingBackground.
                    BigData[casId.NowBouncing].obj as
                    CwaSpriteSheetSelector;
                bounce.Update(gameTime);
                bounce.Update(ref _ScrollingBackground.BigData[casId.
                    NowBouncing].SourceRectangle);
}
```

6.3 Draw

1. Option: Check if animation is visible
2. Draw animation(id)

Draw:

```
if (_ScrollingBackground.IsVisibleCalc(casId.NowBouncing))
{
                _ScrollingBackground.Draw(casId.NowBouncing);
}
```

7 Clickable Item

An example with a texture which changes when touched.

7.1 Create

```
GroupClickable Group = new GroupClickable();
Point poscPmShe = new Point(...);

CommonSb.UpdateItem_Input UpdateItem_InputShe = new CommonSb.
    UpdateItem_Input();
UpdateItem_InputShe.obj = Group;
UpdateItem_InputShe.tex = Group.Result;
UpdateItem_InputShe.centerRotation = Group.Result.Bounds.Center;
UpdateItem_InputShe.CreateClickInfo = true;
UpdateItem_InputShe.id = casId.she;

UpdateItem_InputShe.pm = poscPmShe.X;
UpdateItem_InputShe.y = poscPmShe.Y;

_ScrollingBackground.Import(UpdateItem_InputShe);
```

7.2 Update

1. Option: Use a class TimeAmounts member like ElapsedCountUpdates to update your item less often
2. Option: Check if item is visible. ScrollingBackground.IsVisibleCalc(casId.she)
3. Get your item from the ScrollingBackground.BigData...
4. Update item

```
if (_ElapsedCountUpdates)
{
if (_ScrollingBackground.IsVisibleCalc(casId.she))
{
GroupClickable She = _ScrollingBackground.BigData[casId.she].obj as
    GroupClickable;
She.Next(ScreenManager, casTextureVariant.Original, Color.White,
    _ElapsedSeconds2, _ElapsedSeconds3, _ElapsedSeconds10);
}
}
```

7.3 Touch

1. ScrollingBackground sets ItemsActionIdBody to the items id. Use a switch to process your item.
2. Cast your item from the ScrollingBackground
3. Test which part of the item was touched and save in ItemsActionIdBodyPart

CalculateItemsActionId(...):

```
switch (_ItemsActionIdBody)
{
case casId.she:
{
GroupClickable Group = _ScrollingBackground.BigData[casId.she].obj as
    GroupClickable;
_ItemsActionIdBodyPart = Group.Test(_vRelativeTexture);

if (_ItemsActionIdBodyPart == casId.Undef)
{
_ItemsActionIdBody = casId.Undef;
}
}
break;

...
}
```

7.4 Draw

```
if (_ScrollingBackground.IsVisibleCalc(casId.she))
{
        _ScrollingBackground.Draw(casId.she);
}
```

8 Functions: Item visibility and calculation

In many cases, if an item is not visible it dont needs to be calculated.

Hidden items still may need to be calculated,
eq. if an item moves from one hidden position to a visible.

- IsVisibleCalc(casId):bool
- Hidden(casId):bool
- CalcHidden(casId):bool //deprecated
- CalcVisible(casId):bool
- CalcVisibleThenUpdate(casid, wHours, gametime, multi):bool
- CalcHiddenCached(id, dt, pm, x, wHours):bool

9 Functions: Positions on screen, timeline

- GetScreenDraw(casId):Point
- GetRotCenterScreen(casId):Point
- GetDateTime(Point screen):DateTime
- GetPosMaybeShifted(casId):Point
- GetTouchPosRelativeToPos(Point):Point
- GetPixelPosOnSlide(Point posScreen):Point
- GetPixelPosOnSlide(DateTime dt) : Point

10 Functions: Items on the timeline

- Import(UpdateItem)
- Draw(casId)
- Draw(UpdateItem) // depr
- RemoveItem(casId)

11 Members:

- UpdateItem: all properties of a background item
- BigData: Dictionary holding all background items, except notes, special case.
- dClickable: Dictionary; All clickable background items are here

12 Closely related projects and classes:

See the class documentation for a better description

- **Project CwaNotes:**
  class Cas.CwaTexNotesManager:
  Manages notes. Load from library, determine which one to show.

- **Project CwaTexture:**
  Very often needed. Used for handling textures.

  class CwaCommon.CwaSpriteSheetSelector
  Spritesheet Animations
  class CwaCommon.MeterPixel
  Used to calculate the lensflare of the sun.

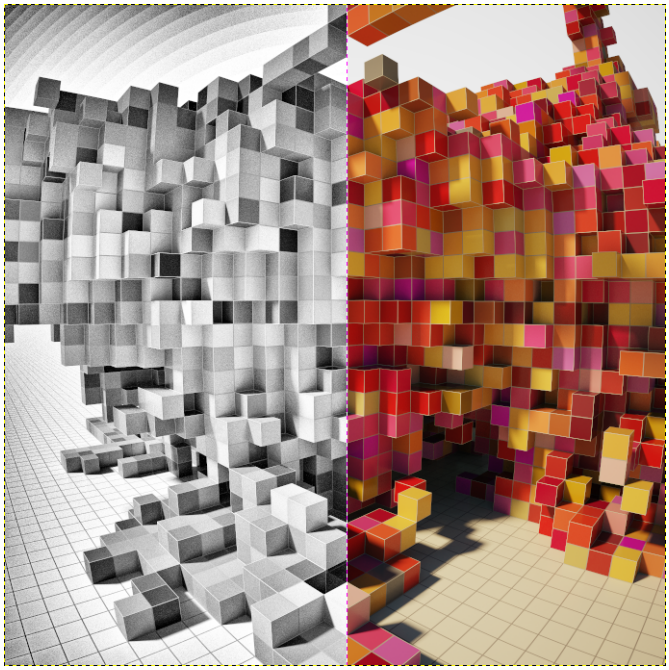
- **Project CwaSunMoonEngine**
  class CwaCommon.SkyEngine
  Use to calculate daylength for the location of your device.

13 Features, Bugs, Ideas

- Load, Delete items from medialibrary is not tested enough. If thousands are loaded and unloaded and the app is running for days, its very likly that a bug will occur. This case was not requested by customer, thus not tested enough.
  If a const. number of items is used testcases succeed.

- Load, Unload, Delete items should be simpler.
- Automatic test for much more items need to be done for Load, Unload, Delete.
- Sb texture should not only fill the whole viewport, but should work as a normal texture in 3d space, which handles touch events. Also memory consumption is an issue, because many instances of Sb will exist, also items need to be shared among those instances, hidden textures need to be stopped etc. That is a lot of work and know how.
  May a simpler solution whould be to have only one texture active and moving and the others just get a screenshot of this.
- Sb has a modi for low memory devices, default mode, full mode. This mode determines if only necessary parts will loaded or also parts like blur effects etc. Currently its hard coded to default mode just to avoid testing the other not used cases.
- Color tint of layers and items is done only for cases. Greyscale, default Color, others not implemented.

14 Feature:



Example: Bulk of random notes/boxes