RhythmTool 1.9 Documentation

Table of Contents

Contact & info

If you have any questions, suggestions, feedback or comments, please do one of the following:

- Send me an Email: tim@hellomeow.net
- Make a post in the RhythmTool thread

Overview

RhythmTool is a straightforward scripting package for Unity, with all the basic functionality for creating games that react to music.

RhythmTool analyzes a song without the need of playing the song at the same time. It can analyze an entire song before playing it, or while it's being played.

There are a number of types of data it provides:

- Beats
- Onsets
- Changes in overall intensity
- volume

This data can be used in various ways and is provided through lists and UnityEvents.

Change log

Version 1.9:

- Added events SongLoaded and SongEnded
- Added support for arbitrary number of channels
- Refactored Analysis and Util classes
- Multiple bug fixes and optimizations

Update Guide

All of RhythmTool's fields and properties have been renamed to match Unity's naming conventions. Every field and property is camelCase. "TotalFrames" has become "totalFrames", "CurrentFrame" has become "currentFrame", "High" has become "high", etc.

The Analysis references in RhythmTool (Low, Mid, High and All) have been replaced with AnalysisData. Instead of float arrays that contain the data, AnalysisData has a number of ReadOnlyCollections. These have the advantage that you can assign them once, and they will get updated with new data for every new song automatically.

Here is an oversimplified example on how to access the magnitude array and how to find an onset in 1.7 and what to change to make it work in versions after 1.8.

Replace this:

```
Analysis all = rhythmTool.All;
float[] magnitude = all.magnitude;
int currentFrame = rhythmTool.CurrentFrame;
Onset onset = all.GetOnset(currentFrame);
```

with this:

```
AnalysisData all = rhythmTool.all;
ReadOnlyCollection magnitude = all.magnitude;
int currentFrame = rhythmTool.currentFrame;
Onset onset = all.GetOnset(currentFrame);
```

Or to find out if a beat, onset or change occurs, you can use events like this:

```
	void Start()
	{
		eventProvider = AddComponent<RhythmEventProvider>();

		//subscribe to an event like this, or use the inspector to subscribe any
		//compatible method like you can with any Unity Event.
		eventProvider.OnOnset.AddListener(OnOnset);
	}

	private void OnOnset(OnsetType type, Onset onset)
	{
		if(type == OnsetType.low)
			Debug.Log("Low Onset Occured: " + onset);
	}
```

With RhythmEventProvider you don't need to have any reference to RhythmTool. You can attach it to any GameObject and RhythmTool will invoke its events when needed.

If you use storeAnalyses, make sure to remove all old .rthm files from the game's persistent datapath, because they're no longer compatible. For Windows this is "C:\Users\User Name\AppData\LocalLow\Company Name\Project Name"

How to use

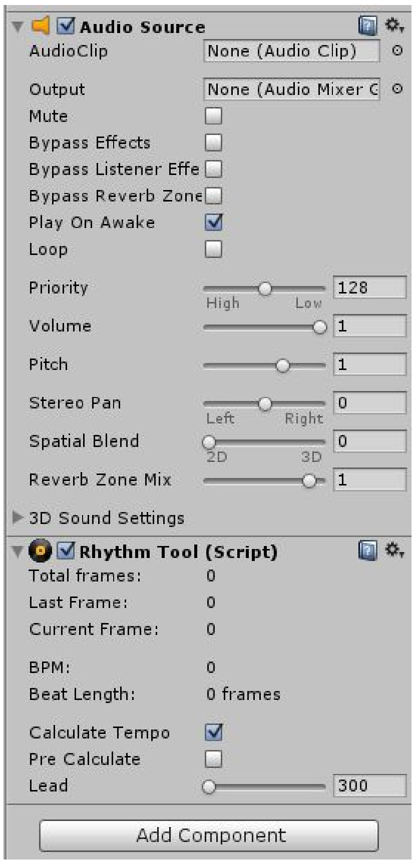
Importing RhythmTool

To begin using RhythmTool, just import the Unity Package. After that, it can be added as a component on a GameObject.

Importing mp3 importer

If you want to use the MP3 importer, download this package and import it into your project.
http://hellomeow.net/files/AudioImporter.zip
The package contains examples on how to both use it with and without RhythmTool as well as some additional instructions.
Important: some of the importers require the api compatibility level to be set to .Net 2.0 in the player settings. Set the api compatibility level to .NET 2.0 if this is not already the case.

Tutorial 1: Basic setup



Audio Source
AudioClip None (Audio Clip)
Output None (Audio Mixer C
Mute ☐
Bypass Effects ☐
Bypass Listener Effe ☐
Bypass Reverb Zone ☐
Play On Awake ☑
Loop ☐
Priority 128 (High – Low)
Volume 1
Pitch 1
Stereo Pan 0 (Left – Right)
Spatial Blend 0 (2D – 3D)
Reverb Zone Mix 1
3D Sound Settings
Rhythm Tool (Script)
Total frames: 0
Last Frame: 0
Current Frame: 0
BPM: 0
Beat Length: 0 frames
Calculate Tempo ☑
Pre Calculate ☐
Lead 300
Add Component

In this tutorial we're going to make a simple script that will play and analyze a song and show the data.

I made a GameObject named "Controller", which will do everything. First, a RhythmTool component has to be added to the Controller. This can be done by using the Add component menu. It's located under "Audio". This will also add an AudioSource.

If you're using Unity 5, you might want make sure "Spatial Blend" is set to 0. Otherwise, the music will be affected by 3D audio calculations like Doppler shift and attenuation.

In the inspector, we can see some information and we can also change some options. By default, Rhythmtool will analyze a song while it is playing, but we can also choose to pre calculate the results for the entire song before playing it and to store this data. We can also change the head start that the analysis will have and whether or not to calculate the song's tempo.

Now RhythmTool is added, we need to give it a song to analyze. To do this, make a new script. I made a script named "BasicController" and declared two fields of RhythmTool and AudioClip.

```
public class BasicController : MonoBehaviour
{
	public RhythmTool rhythmTool;
	public AudioClip audioClip;
```

Now we can either drag and drop the RhythmTool component in the inspector, or use GetComponent to assign it. In this case, I'm using GetComponent for rhythmTool and the inspector for audioClip.

```
	void Start ()
	{
		rhythmTool = GetComponent<RhythmTool>();
	}
```

Now we can give it an AudioClip of a song, with RhythmTool.NewSong().

```
        void Start ()
        {
                rhythmTool = GetComponent<RhythmTool>();
                rhythmTool.NewSong(audioClip);
                rhythmTool.SongLoaded += OnSongLoaded;
        }
```

After calling RhythmTool.NewSong(), it will raise an event when it's ready to play the song. If it's set to pre calculate, this will happen when it's done analyzing the song, or when it's done loading the previously stored data. Otherwise, it will be almost instantaneous.

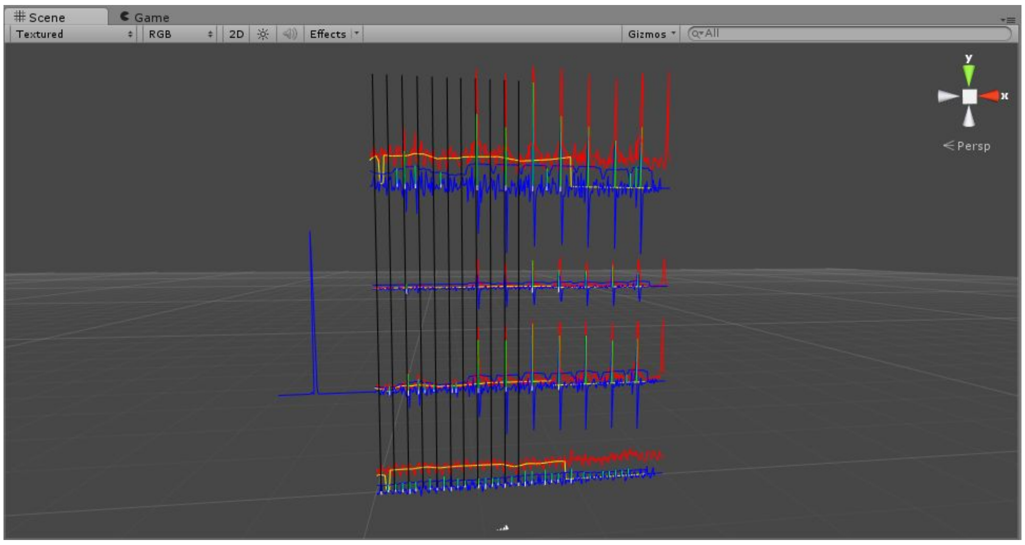
The event that's being raised and that we have to subscribe to, is called “SongLoaded”. When RhythmTool is ready, we want it to start playing and analyzing the song.

```
        void OnSongLoaded()
        {
            rhythmTool.Play();
        }
```

That’s basically it. To get a look at the data, we can call rhythmTool.DrawDebugLines() in Update(). DrawDebugLines can be a bit performance intensive.

```
        void Update ()
        {
                rhythmTool.DrawDebugLines ();
        }
```

This will show some graphs that represent the different results. It will scroll towards the left side, which is the current time in the song. On the right side we can see new data being added. See tutorial 4 for more information about this data.



Tutorial 2: using the importer

Now we have a controller that will take an AudioClip, analyze it and show the data, it would be nice if we could choose a song in game. The importer takes care of this. The importer is a separate, free package that contains a file browser and three different importers.

- NAudioImporter, which uses NAudio, which is licensed under the Ms-PL. It might require licensing for using MP3 technology.
- BassImporter, which uses Bass.dll and Bass.net, which require separate licensing for commercial projects. This importer is fast and can use the OS's mp3 decoder, so no MP3 license is needed.
- MobileImporter, which is for mobile platforms only, because Unity's WWW class will stream mp3 files on mobile platforms only.

For this tutorial i'm going to modify the script from the previous tutorial and use NAudioImporter. To begin using the importer, just add an NAudioImporter component to the controller GameObject from the previous tutorial. It's also located under Audio.

N Audio Importer (Script)
Script NAudioImporte

We can remove the AudioClip field from the previous tutorial, since it's no longer needed. Add NlayerImporter and Browser instead.

```
public class NaudioExample : MonoBehaviour
{
	public NAudioImporter importer;
	public AudioSource audioSource;
	private Browser browser;
```

Now we have to assign it. It can be done by dragging and dropping it in the inspector, or with GetComponent(). Next we also want to open the browser. Browser.Create() will do this by instantiating a prefab. You can also add the prefab to the scene and drag the Browser component in the inspector.

```
	void Start()
	{
		rhythmTool = GetComponent<RhythmTool>();
		browser = Browser.Create(gameObject);
	}
```

Next we have to subscribe a method to the browser's FileSelected event. This event occurs whenever a file is clicked in the browser. It passes along a string with the path of the clicked file.

```
void Start()
{
    rhythmTool = GetComponent<RhythmTool>();
    browser = Browser.Create();
    browser.FileSelected += OnFileSelected;
}
```

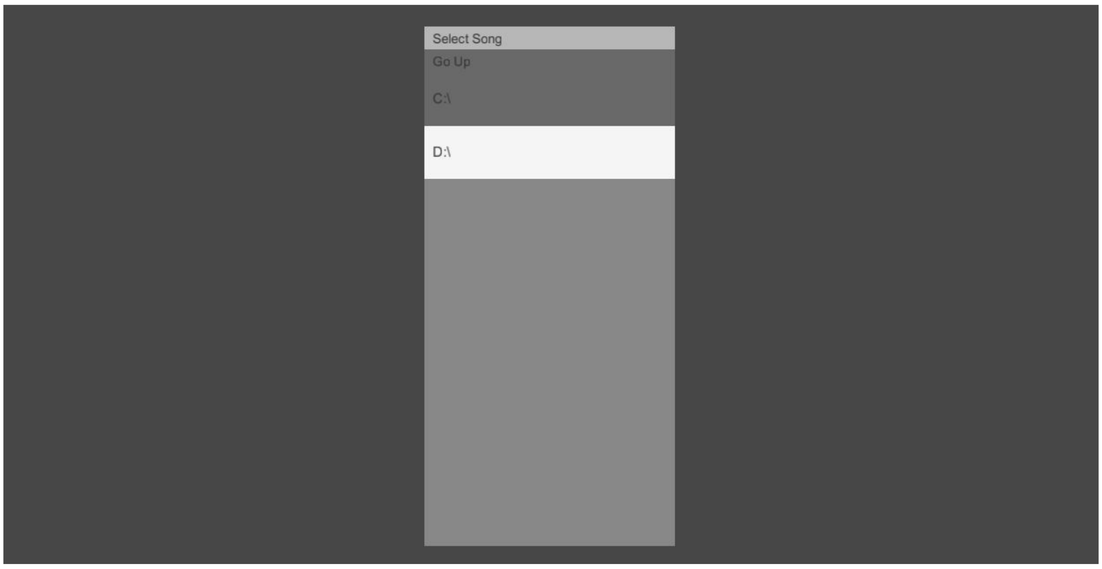
The importer can import the entire file at once, but it can also import it while it’s being played. ImportFileStreaming() does this. In this case we want it to import 20 seconds of the song initially, otherwise RhythmTool will start analyzing parts of the song that haven’t been imported yet, resulting in no data at all.

```
        public void OnFileSelected(string path)
        {
                importer.ImportFileStreaming(path,20);
        }
```

Because decoding an MP3 file can take a while, it’s being done on a separate thread. The importer has an event for when the AudioClip when it’s ready. The event we have to subscribe to is “FileLoaded”. Once we get the AudioClip, we can give it to RhythmTool.

```
        public void OnFileLoaded(AudioClip clip)
        {
                rhythmTool.NewSong(clip);
        }
```

That’s all. Now we can select a song in game and it will be imported and analyzed.



Tutorial 3: using Events

The easiest and most straightforward way to use RhythmTool's results, is by using the RhythmEventProvider component. This component, as the name suggests, provides UnityEvents related to RhythmTool. It can be added to any GameObject and RhythmTool will invoke it's event's. These are UnityEvents , which work slightly different from C#'s events

The following events are available:

- onBeat(Beat beat)
  Whenever a beat occurs, this get's called. The Beat that's passed along contains the index, length and most probable BPM at the time.
- onSubBeat(Beat beat, int count)
  This is the same as onBeat, but is called every 1/4th of a beat. "count" ranges from 0 to 3 and indicates which quarter of the beat occured. 0 is the first, 1 is the second, etc.
- onOnset(OnsetType type, Onset onset)
  This gets called whenever an onset occured. "type" indicates in which frequency range the onset was found. "onset" includes the index, strength and rank of the onset.
- onChange(int index, float change)
  Whenever a change occures, this is called. The index and magnitude are passed.
- onFrameChanged(int currentFrame, int lastFrame)
  This is called when currentFrame changes. It's also called for each frame that's skipped. It passes the current frame and the last analyzed frame index.
- TimingUpdate(int currentFrame, float interpolation, float beatLength, float beatTime)
  This is called every Update and includes information that's useful for interpolating in between frames and beats.
- onNewSong(string name, int totalFrames)
  This is called when RhythmTool has finished loading a new song. It passes the song's name and the song's length in frames.
- onSongEnded()
  This is called when the song has ended.

In this tutorial we're going to make some cubes react to the beat. We can build upon the script from tutorial 1. First, add a RhythmEventProvider component to the controller GameObject. It's also located under Audio in the components menu. Also add two cubes to the scene and place them anywhere in the camera's field of view.

Next, add fields for the two cube's Transforms and the RhythmEventProvider to the script.

```
public class EventsController : MonoBehaviour
{
        public RhythmEventProvider eventProvider;
        public Transform cube1Transform;
        public Transform cube2Transform;
```

You can drag and drop the cubes and the RhythmEventProvider component in the inspector to assign them. I use GetComponent() instead for the RhythmEventProvider.

Next we have to make some methods to subscribe to the events. The first cube is going to react to every beat and the second cube is going to react to every half beat. The cubes will get a random scale every time their event is invoked.

```
	void OnBeat(Beat beat)
	{
		cube1Transform.localScale = Random.insideUnitSphere;
	}

	void OnSubBeat(Beat beat, int count)
	{
		if(count == 0 || count == 2)
			cube2Transform.localScale = Random.insideUnitSphere;
	}
```

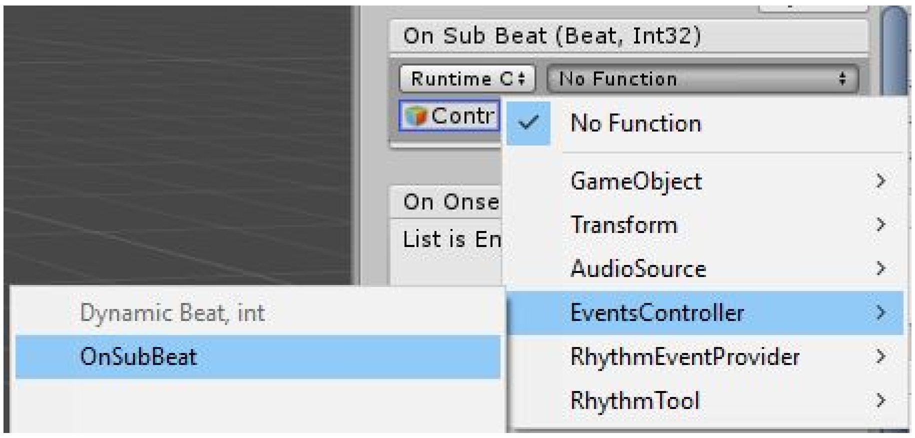
The two methods should match the Event's signature. OnBeat should have a parameter for a Beat and OnSubBeat should have parameters for a Beat and an int. Now we can subscribe our methods to the Events with AddListener().

```
	void Start()
	{
		rhythmTool = GetComponent<RhythmTool>();
		eventProvider = GetComponent<RhythmEventProvider>();

		eventProvider.onBeat.AddListener(OnBeat);
		eventProvider.onSubBeat.AddListener(OnSubBeat);

		rhythmTool.NewSong(audioClip);
	}
```

It's also possible to subscribe to events in the inspector, just like with any other UnityEvent. In that case, the method's can also have no parameters, but they have to be public.



And that's it. If you hit play, the cubes should react to the beat.

Tutorial 4: Using AnalysisData

RhythmTool has three main types of data:

- Beats
  These represent the rhythm of the song.
- AnalysisData
  These contain data and detected peaks (onsets) of different frequency ranges.
- Changes
  These represent changes in the overall intensity of the song and are useful for telling segments of the song apart.

For each 0.03333 seconds, or 30 times per second, RhythmTool analyzes and stores some data, which can then be synchronized with the song. The data can be viewed as being stored in frames. Each piece of data has in index, which corresponds with it's time in the song.

In this tutorial, we're going to start with AnalysisData. AnalysisData contains the results of an Analysis, which are Lists with the following information:

- magnitude
  The combined magnitude of all frequencies in the specified frequency range. Basically the loudness.
- magnitudeSmooth
  A smoothed variant of the above.
- magnitudeAvg
  A value that interpolates from trough to trough and peak to peak. This is like a smooth version of magnitude, but without big variations.
- flux
  The difference between this frame's and the previous frame's magnitudes.
- onsets
  The beginning of a note, sound or beat.

There are 4 default AnalysisDatas. low, mid, high and all. For now we're going to look at low. To access this data, declare an AnalysisData and assign one of the AnalysisDatas from rhythmTool to it.

```
	private AnalysisData low;

	void Start ()
	{
		rhythmTool=GetComponent<RhythmTool>();
		low = rhythmTool.low;
		rhythmTool.NewSong(audioClip);
		...
```

low looks at the lower frequencies in the song, so it will detect the loudness and onsets of things like kicks. How do we use it? As an example in this tutorial, we're going to make some lines to represent some of the data.

An important variable that RhythmTool provides is currentFrame. currentFrame is the index of the frame that belongs to the point in the song that's currently being played. With currentFrame you can find which data to use and when to use it.

With a loop, we are going to look at the frame corresponding with the current time in the song, and the 100 frames after that. We're going to draw a line representing the loudness. The horizontal position of this line is determined by the index of this frame and the current time in the song, so the lines will appear to move towards a point, and meet that point when their time in the song arrives.

```
	void Update ()
	{
		//Don't do anything if there is no song loaded.
		if (!rhythmTool.songLoaded)
		return;

		for(int i = 0; i < 100; i++)
		{
			//make sure we don't try to access a frame that's out of range
			int frameIndex = Mathf.Min(rhythmTool.currentFrame + i,
			rhythmTool.totalFrames);

			Vector3 start = new Vector3(i, low.magnitude[frameIndex], 0);
			Vector3 end = new Vector3(i+1, low.magnitude[frameIndex+1], 0);
			Debug.DrawLine(start, end, Color.black);
```

When there is an onset, draw a red line

```
			float onset = low.GetOnset(frameIndex);

			if (onset>0)
			{
				start = new Vector3(x,0,0);
				end = new Vector3(x,onset,0);
				Debug.DrawLine(start,end,Color.red);
			}
```

Next to the data about onsets and loudness, RhythmTool also tries to detect a song's tempo.

Instead of looking for relatively loud peaks, this method looks at larger portions of the song and tries to detect repetition. This is a more reliable way of detecting a beat most of the time. Onsets can be detected for any sudden peak, while this looks for repetition in the song.

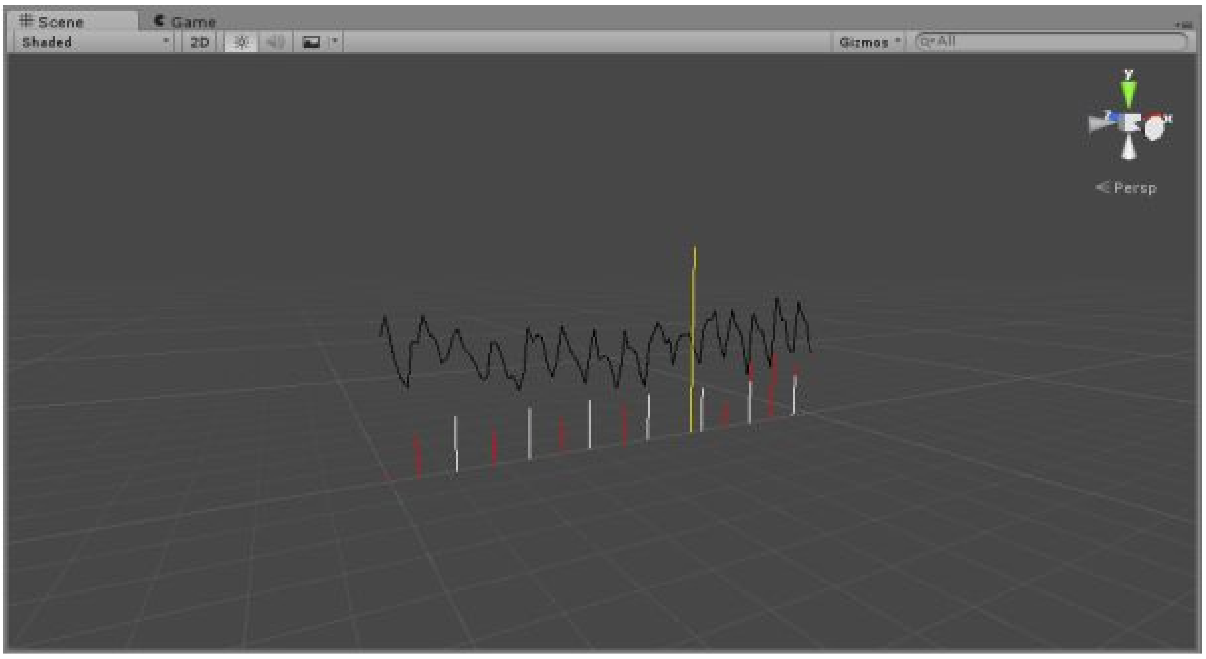
With RhythmTool.IsBeat(), we can see if a beat should occur at an index. IsBeat() returns 1 if a whole beat occurs, 2 if a half beat occurs and 4 if a quarter beat occurs.

```
if(rhythmTool.IsBeat(frameIndex) == 1)
{
        start = new Vector3(x,0,0);
        end = new Vector3(x,10,0);
        Debug.DrawLine(start,end,Color.white);
}
```

RhythmTool can also look for and detect larger changes in the song based on loudness. Relatively big increases or decreases in overall loudness are a good way of telling different sections of a song apart.

```
if(rhythmTool.IsChange(frameIndex))
{
        float change = rhythmTool.NextChange(frameIndex);
        start = new Vector3(i, 0, 0);
        end = new Vector3(x, Mathf.Abs(change), 0);
        Debug.DrawLine(start, end, Color.yellow);
}
```

At this point, it should look like this. There is a black line representing the loudness, red lines showing upcoming detected onsets, white lines showing upcoming beats and yellow lines that show upcoming changes in the song.



Scene
Game
Shaded
2D
Gizmos
All
Persp