



Unityはじめるよ

～ Unity2018のスク립タブルレンダーパイプラインを試してみた～

統合開発環境を内蔵したゲームエンジン
<http://japan.unity3d.com/>

※いろんな職業の方が見る資料なので説明を簡単にしてある部分があります。正確には本来の意味と違いますが上記理由のためです。ご了承ください。
この資料内の一部の画像、一部の文章はUnity公式サイトから引用しています。

資料の内容

- ・ スクリプタブルレンダーパープラインって？
- ・ 使い方
- ・ 使ってみて

スクリプトブルレンダーパイプラインって？

スクリプタブルレンダーパイプラインって？

レンダーパイプラインとは、レンダリングパイプライン、グラフィックスパイプラインともいう
オブジェクトを画面に描画するまでの様々な工程（カリング、各種座標変換、陰影計算、ラスタライズなど）を表す言葉。

一昔前はハードウェアに強く依存していた。
※ハードの性能で描画の綺麗さが決まっていた

今は描画の綺麗さはソフトウェアに依存するところが大きく、ハードは処理の速さに影響を与えることが多い。

Unityでは、
その様々な工程の部分がブラックボックスになっており、
Unityが提供するパイプラインを利用するしかなかった。

レンダリング手法として、下記が提供されていた。

- **バーテックスリットレンダリング**
→処理は軽いが表現力が乏しい。
- **フォワードレンダリング**
→表現力は豊かだが、ライトの数が増えるに比例して処理
負荷が増えていく。
- **デファードレンダリング**
→ライトの数が増えても処理負荷は増えない。表現力も豊
かだが半透明が使えない。

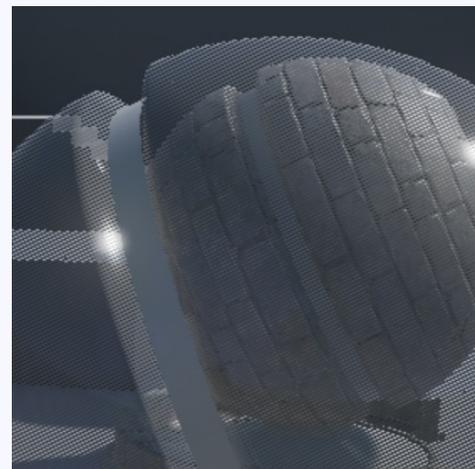
ちょっと小話

PS4 やPCゲームだと、デファードレンダリングを採用しているゲームが多い。

理由は、凝ったライティングでも処理負荷が安定するため。

ただ、デファードレンダリングでは仕組み上、半透明処理ができないため、半透明処理に**ディザリング**を使ったり、半透明部分だけフォワードレンダリング組み合わせたレンダリング手法を使うなどの工夫が見られる。

モンハンやFFでも
ディザリングが使われているのが
確認できた。



ディザリングの例
[もんしよの巣穴blog](#)様から引用

Unity5から試験的に

「レンダリングの工程を自分で構築する仕組み」

が提供されるようになり、Unity2018から正式版として組み込まれることとなった。

今までブラックボックスだった部分が、自分で組めるようになったのだ。

これを**スクリプタブルレンダーパイプライン**という。

※以下**SRP**

SRPのメリット/デメリット

メリット

表現力と処理速度の天秤を、
開発者がコントロールできるようになった。

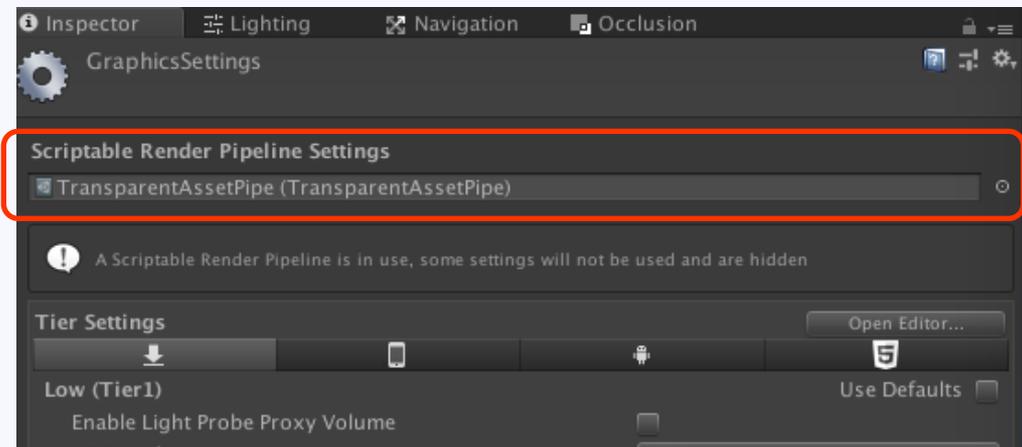
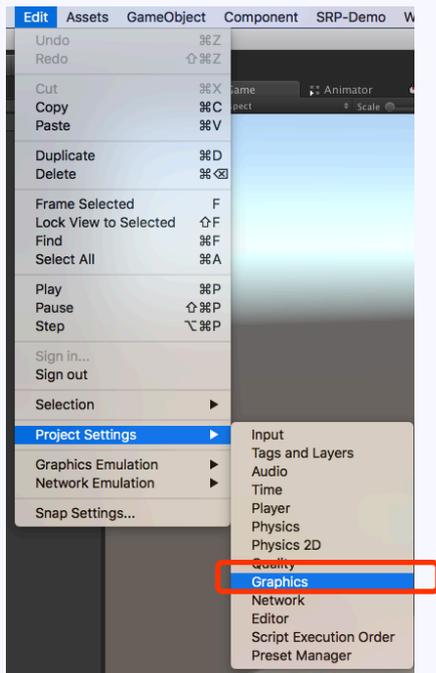
デメリット

学習コストが高い（つまり難しい）

使い方

使い方（というか試してみた方法）

スクリプタブルレンダーパイプラインを使うには、**Edit > ProjectSettings > Graphics** の **Scriptable Render Pipeline Settings** に **Render Pipeline Asset** をセットする。



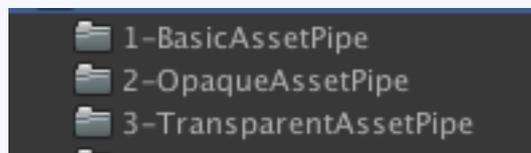
Render Pipeline Asset

はどうやって準備するの？

まずは、
デモプロジェクトに含まれるアセットを使わせてもらおう。

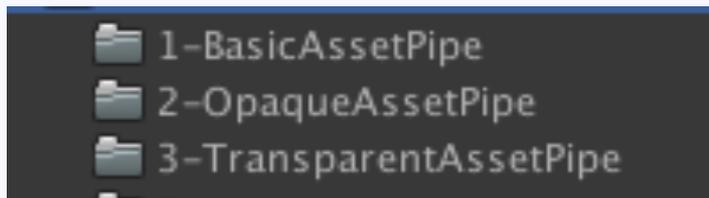
デモプロジェクト：

<https://github.com/stramit/SRPBlog/tree/master/SRP-Demo>



この中に3種類の**Render Pipeline Asset**が含まれている

各アセットの説明



- **BasicAssetPipe**
→画面をクリアするだけ
- **OpaqueAssetPipe**
→不透明オブジェクトの描画が行える
- **TransparentAssetPipe**
→不透明／透明のオブジェクトの描画が行える

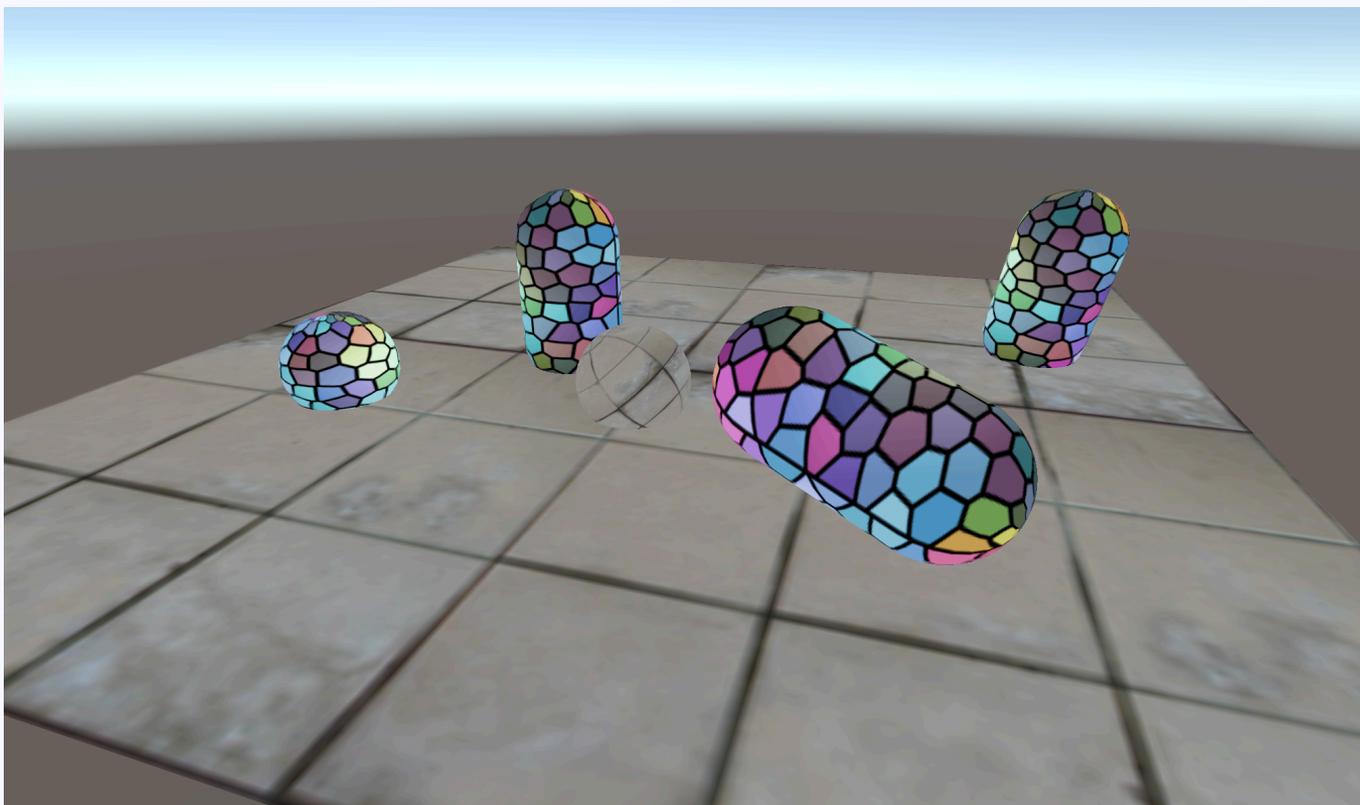
とりあえずデモを試してみる

- **BasicAssetPipe**



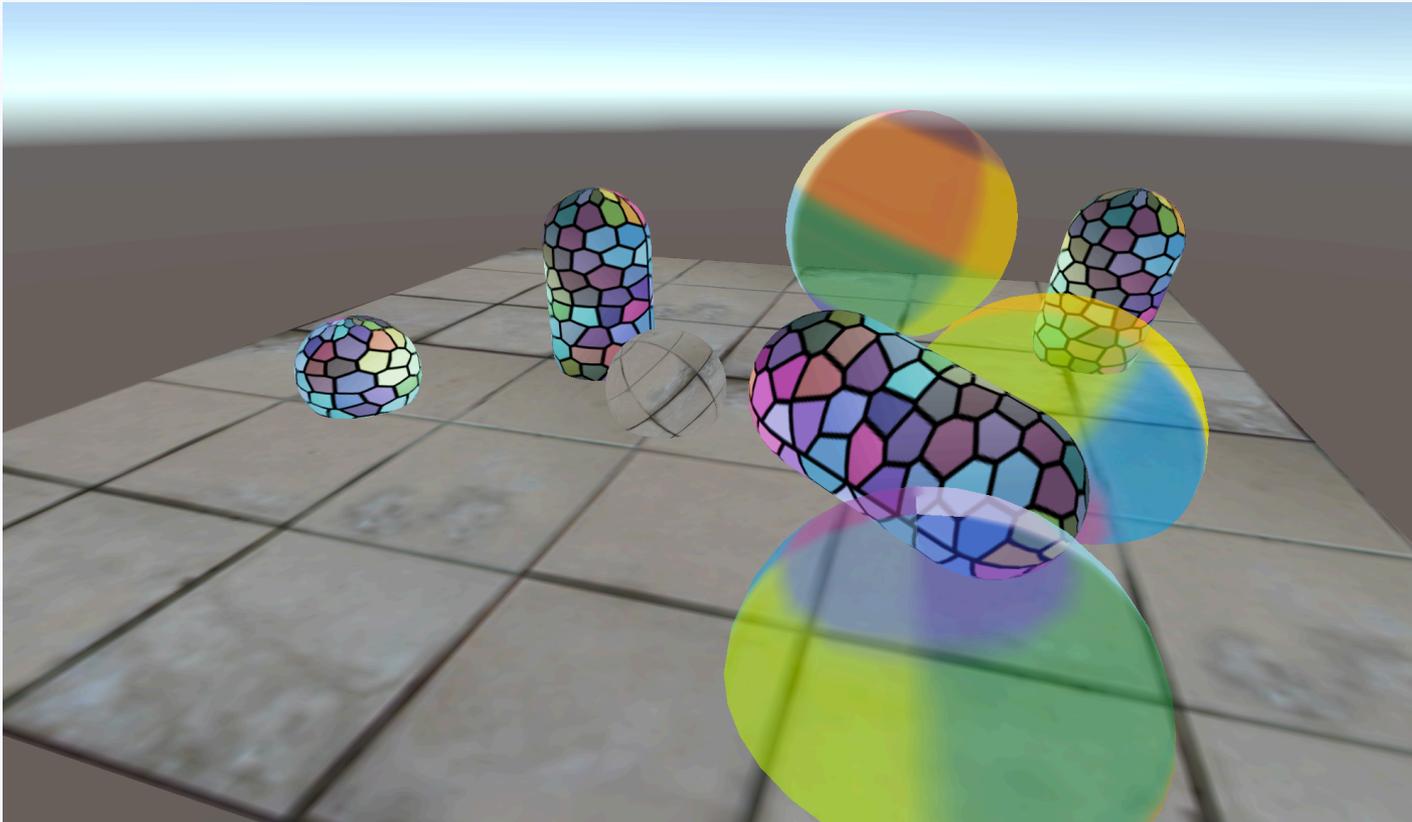
とりあえずデモを見てみる

- OpaqueAssetPipe



とりあえずデモをしてみる

- TransparentAssetPipe



TransparentAssetPipeの中身を見る

```
public override void Render(ScriptableRenderContext context, Camera[] cameras)
{
    base.Render(context, cameras);

    foreach (var camera in cameras)
    {
        // Culling
        ScriptableCullingParameters cullingParams;
        if (!CullResults.GetCullingParameters(camera, out cullingParams))
            continue;

        CullResults cull = CullResults.Cull(ref cullingParams, context);

        // Setup camera for rendering (sets render target, view/projection matrices and other
        // per-camera built-in shader variables).
        context.SetupCameraProperties(camera);

        // clear depth buffer
        var cmd = new CommandBuffer();
        cmd.ClearRenderTarget(true, false, Color.black);
        context.ExecuteCommandBuffer(cmd);
        cmd.Release();

        // Draw opaque objects using BasicPass shader pass
        var settings = new DrawRendererSettings(camera, new ShaderPassName("BasicPass"));
        settings.sorting.flags = SortFlags.CommonOpaque;

        var filterSettings = new FilterRenderersSettings(true) { renderQueueRange = RenderQueueRange.opaque };
        context.DrawRenderers(cull.visibleRenderers, ref settings, filterSettings);

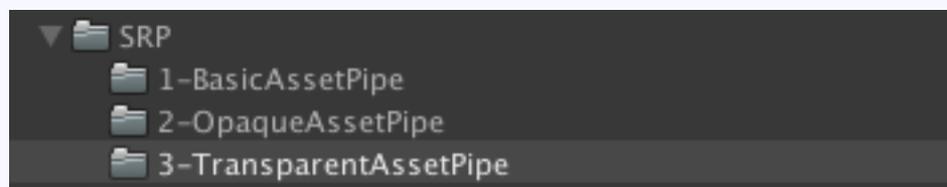
        // Draw skybox
        context.DrawSkybox(camera);

        // Draw transparent objects using BasicPass shader pass
        settings.sorting.flags = SortFlags.CommonTransparent;
        filterSettings.renderQueueRange = RenderQueueRange.transparent;
        context.DrawRenderers(cull.visibleRenderers, ref settings, filterSettings);

        context.Submit();
    }
}
```

他のプロジェクトで使うには？

デモプロジェクトの「**SRPフォルダ以下**」をコピーして他のプロジェクトに貼り付ける



先ほどの手順で、RenderPipelineAssetをセットすると…

SRPに対応しているシェーダーを使ったマテリアルなら
画面に描画される。

それ以外は描画されない・・・。

シェーダーをSRPに対応させる方法

デモプロジェクトの場合、シェーダー内の**Tags**の中に、

”LightMode”=”BasicPass”

と書くと対応できた。

```
Tags {"LightMode" = "BasicPass"}
```

“BasicPass”の部分は、使う RenderPipelineAssetにより変える。

シェーダーの対応を繰り返しているうちにそれっぽい画面になってきた。
しかし、ライティングと影の出し方がわからん・・・。



デモプロジェクトだと謎が多いので、 ちゃんとソース一式をプロジェクトに入れてみる

凹みTipsさんが、

「Unity 本体とのバージョンと強く結びついているため、それぞれのリリースにあったバージョンの Unity で動かす必要があります。」

と書いている通り、
しっかりバージョンを合わせないと動きませんでした。

UnityのバージョンとScriptableRenderPipelineのバージョンをしっかりと揃えて、Githubから取得しました。

↓↓ GithubからScriptableRenderPipelineを持ってくる方法↓↓ ----

```
git clone https://github.com/Unity-Technologies/ScriptableRenderPipeline  
cd ScriptableRenderPipeline  
git checkout Unity-2018.1.0b6  
git submodule update --init --recursive --remote
```

----※凹みTipsさんから引用

その後ダウンロードした一式をプロジェクトにぶっこむ。

しかし、

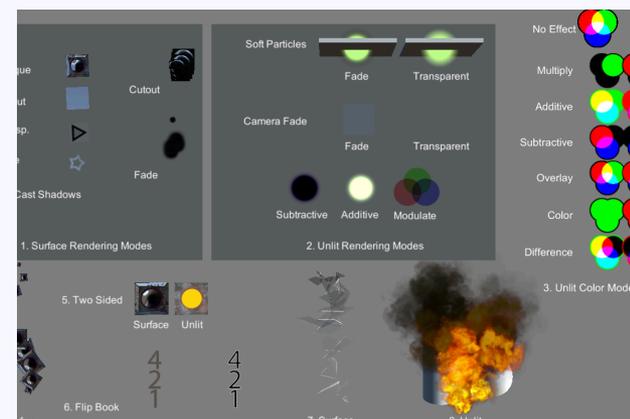
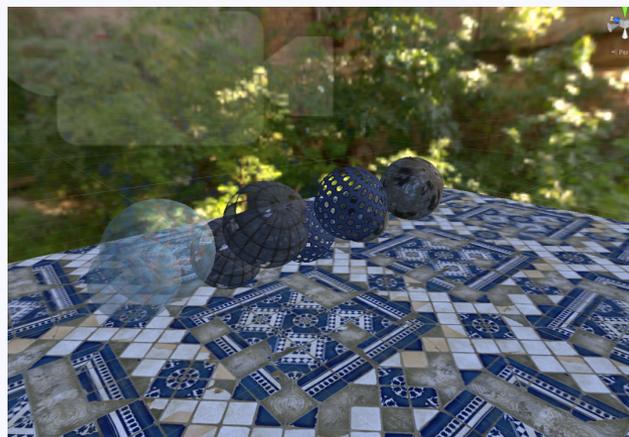
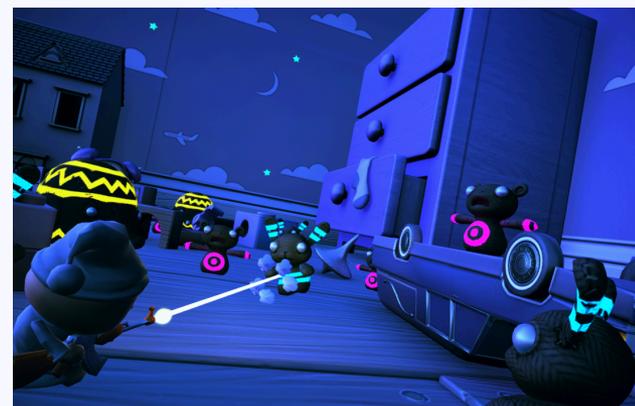
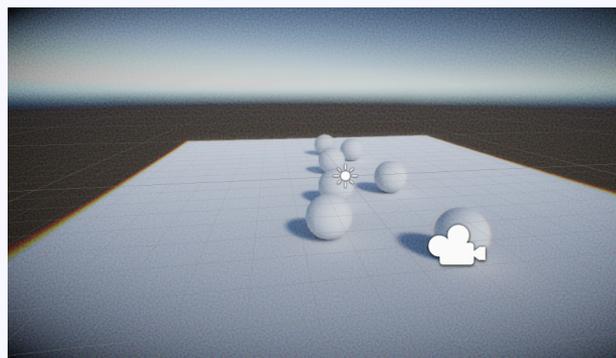
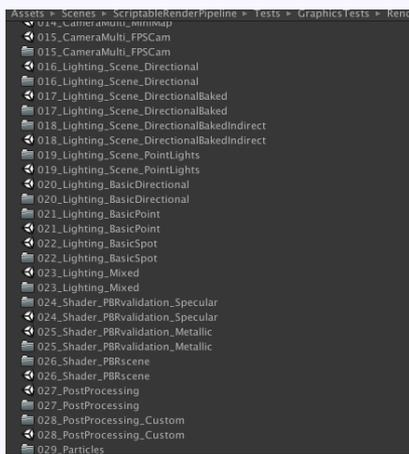
Unsafe code requires the `unsafe` command line option to be specified

というエラーが！！

unsafeなコードを許可する必要があると。
その方法は、

拡張子が「.asmdef」なるファイルが何箇所かにあるので、
上記のエラーが出ているスクリプトの階層においてある
「.asmdef」を選択し、Allow 'unsage' Code のチェックを入れ
Applyエラーが出なくなるまで繰り返せばOK。

これでめでたく、ScriptableRenderPipelineが使えるようになります。サンプルも沢山入っていました。



Unity標準のパイプライン(Foward)と
LightWeightRenerPipeline(LWRP)を
実機で動作速度比較してみました。

検証機種 iPhone6

条件

```
Batches: 644      Saved by batching: 179  
Tris: 64.9k      Verts: 96.1k  
Screen: 662x1177 - 8.9 MB  
SetPass calls: 204  Shadow casters: 0  
Visible skinned meshes: 301  Animations: 0
```

※LWRPの方は、シェーダーを完全に直していないので、
色が一部変ですが、ほぼ同じ条件です。

結果

Unity標準



45fps

LWRP



59fps

軽い！ 軽いよLWRP！

しかもリアルタイムシャドウの描画の方法が違うのか、影の描画がやわらかくて綺麗。LWRPの方が影の解像度を低くしているのに。



Unity標準



LWRP

まとめ

まとめ

調べてみるまでは、正直もっと手軽なものだと思っていた。

現時点ではまだまだ情報が少ないと感じる。構造の難しさだけじゃなくて調べるコストも大きい。

進化速度が速いのか（すぐに情報が古くなる）、調べた通りにやってもうまく動かなかったりと、一筋縄じゃいかない。もう少し情報が出揃ってくるまでは、手を出すのに覚悟がいる。

ただ、モバイル開発者には美味しい話ではある。

参考サイト

Unity : <https://unity3d.com/jp/srp>

UnityBlog : <https://blogs.unity3d.com/jp/2018/01/31/srp-overview/>

デモプロジェクト :

<https://github.com/stramit/SRPBlog/tree/master/SRP-Demo>

マニュアル :

<http://resetoter.cn/UnityDoc/Manual/ScriptableRenderPipeline.html>

ソース一式(LightWeight) :

<https://github.com/Unity-Technologies/ScriptableRenderPipeline/tree/master/com.unity.render-pipelines.lightweight/LWRP>

Unityフォーラム :

<https://forum.unity.com/threads/feedback-wanted-lightweight-render-pipeline.518267/>

<https://forum.unity.com/threads/scriptable-render-loop-with-single-pass-forward-lighting.473568/>

凹みTips様 : <http://tips.hecomi.com/entry/2018/02/19/000846>

ロジカルビート様 : <http://logicalbeat.jp/blog/1112/>

土屋つかさのテクノロジーは今か無しか様 : http://d.hatena.ne.jp/t_tutiya/20180327/1522145672

ご清聴ありがとうございました