**Abstract**

Image thresholding is a common task in many computer vision and graphics applications.

画像のしきい値処理は、多くのコンピュータービジョンおよびグラフィックスアプリケーションで一般的なタスクです。

The goal ofthresholding an image is to classify pixels as either “dark” or “light”.

画像をしきい値処理する目的は、ピクセルを「暗い」または「明るい」に分類することです。

Adaptive thresholding is a form of thresholding that takes into account spatial variations in illumination.

適応しきい値処理は、照明の空間的変動を考慮したしきい値処理の形式です。

We present a technique for real-time adaptive thresholding using the integral image of the input.

入力の積分画像を使用したリアルタイム適応しきい値処理の手法を提示します。

Our technique is an extension of a previous method.

私たちのテクニックは、以前のメソッドの拡張です。

However, our solution is more robust to illumination changes in the image.

ただし、このソリューションは、画像の照明の変化に対してより堅牢です。

Additionally, our method is simple and easy to implement.

Our technique is suitable for processing live video streams at a real-time frame-rate, making it a valuable tool for interactive applications such as augmented reality.

我々の技術は、リアルタイムのフレームレートでライブビデオストリームを処理するのに適しており、拡張現実などのインタラクティブなアプリケーションに役立つツールです

Source code is available online.

ソースコードはオンラインで入手できます。

**Introduction**

Image thresholding segments a digital image based on a certain characteristic of the pixels (for example, intensity value).

画像のしきい値処理は、ピクセルの特定の特性（強度値など）に基づいてデジタル画像をセグメント化します。

The goal is to create a binary representation of the image, classifying each pixel into one of two categories, such as “dark” or “light”.

目標は、画像のバイナリ表現を作成し、各ピクセルを「暗い」または「明るい」などの2つのカテゴリのいずれかに分類することです。

This is a common task in many image processing applications, and some computer graphics applications.

これは、多くの画像処理アプリケーションおよび一部のコンピューターグラフィックスアプリケーションで一般的なタスクです。

For example, it is often one of the first steps in marker-based augmented reality systems [Billinghurst et al. 2001; Bradley and Roth 2004; Fiala 2005], and it has been used in high dynamic range photography [Ward 2003].

たとえば、それは多くの場合、マーカーベースの拡張現実システムの最初のステップの1つです[Billinghurst et al。 2001; Bradley and Roth 2004; Fiala 2005]、および高ダイナミックレンジ写真で使用されています[2003年]。

The most basic thresholding method is to choose a fixed threshold value and compare each pixel to that value.

最も基本的なしきい値設定方法は、固定しきい値を選択し、各ピクセルをその値と比較することです。

These techniques have been described and evaluated extensively in a number of survey papers [Weszka and Rosenfeld 1978; Palumbo et al. 1986; Sahoo et al. 1988; Lee et al. 1990; Glasbey 1993; Trier and Jain 1995; Sezgin and Sankur 2004].

これらの手法は、多くの調査論文で広く説明され評価されてきました[Weszka and Rosenfeld 1978; Palumbo et al。 1986; Sahoo et al。 1988;リーら。 1990;グラスベイ1993;トリーアとジャイン1995; Sezgin and Sankur 2004]。

However, fixed thresholding often fails if the illumination varies spatially in the image or over

time in a video stream.

ただし、照明が画像内で空間的に変化したり、ビデオストリーム内で時間とともに変化したりする場合、固定しきい値はしばしば失敗します。

In order to account for variations in illumination, the common solution is adaptive thresholding.

照明の変動を考慮した一般的な解決策は、適応しきい値処理です。

The main difference here is that a different threshold value is computed for each pixel in the image.

ここでの主な違いは、画像内のピクセルごとに異なるしきい値が計算されることです。

This technique provides more robustness to changes in illumination.

この手法により、照明の変化に対する堅牢性が高まります。

A number of adaptive thresholding methods exist [White and Rohrer 1983; Bernsen

1986; Parker 1991; Wellner 1993; Yang et al. 1994; Shen and Ip 1997; Chan et al. 1998; Savakis 1998; Sauvola and Pietikainen 2000; Yang and Yan 2000].

多数の適応しきい値設定方法が存在します[White and Rohrer 1983;バーンセン

1986;パーカー1991; Wellner 1993;ヤンら。 1994; Shen and Ip 1997;チャンら。 1998;サバキス1998; Sauvola and Pietikainen 2000;ヤンとヤン2000]。

Further examples and comparisons can be found in [Venkateswarlu and Boyle 1995; Sezgin and Sankur 2004].

We present a very simple and clear technique using integral images.

私たちの提案は、積分画像を使用した非常にシンプルで明確な手法を紹介します。

Our method is easy to implement for real-time performance on a live video stream.

この方法は、ライブビデオストリームでリアルタイムパフォーマンスを実現するために簡単に実装できます。

Though our technique is an extension to a previous method [Wellner 1993], we increase robustness to strong illumination changes.

この手法は以前の方法[Wellner 1993]の拡張ですが、強い照明の変化に対する堅牢性を高めます。

In addition, we present a clear and tidy solution without increasing the complexity of the implementation.

さらに、実装の複雑さを増すことなく、明確で整然としたソリューションを提供します。

Our technique is also similar to the thresholding method of White and Rohrer for optical character recognition [White and Rohrer 1983], however we present an implementation designed for real-time video.

この手法は、光学式文字認識用のWhite and Rohrerのしきい値処理方法[White and Rohrer 1983]にも似ていますが、リアルタイムビデオ用に設計された実装を提示します。

The motivation for this work is finding fiducials in augmented reality applications.

この作業の動機は、拡張現実アプリケーションで基準を見つけることです。

Pintaric also presents an adaptive thresholding algorithm specifically for augmented reality markers [Pintaric 2003], however his method requires that a fiducial has been located in a previous frame in order for the technique to threshold correctly.

Pintaricは、拡張現実マーカー専用の適応しきい値アルゴリズム[Pintaric 2003]も提示しますが、彼の方法では、この手法でしきい値を正しく設定するために基準が前のフレームに配置されている必要があります。

Our algorithm makes no assumptions and is more general, suitable for use in any application.

私たちのアルゴリズムは仮定をせず、より一般的で、あらゆるアプリケーションでの使用に適しています。

The source code is available online at the address listed at the end of this paper.

ソースコードは、このペーパーの最後に記載されているアドレスからオンラインで入手できます。

要約

照明の変動を考慮した一般的な解決策は、適応的閾値処理です。

大域的閾値処理とは、画像内ピクセルごとに異なる閾値が計算される点が異なります。

私たちが提案する方法は、Wellnerの拡張版で、照明に対して高い堅牢性を発揮する。

**Background**

**Real-time adaptive Thresholding**

In this paper we focus on adaptively thresholding images from a live video stream.

この論文では、ライブビデオストリームの画像を適応的にしきい値処理することに焦点を当てています。

In order to maintain real-time performance, the thresholding algorithm must be limited to a small constant number of iterations through each image.

リアルタイムのパフォーマンスを維持するために、しきい値処理アルゴリズムは、各イメージの小さな一定の反復回数に制限する必要があります。

Thresholding is often a sub-task that makes up part of a larger process.

多くの場合、しきい値設定は、より大きなプロセスの一部を構成するサブタスクです。

For instance in augmented reality, input images must be segmented to locate known markers in the scene that are used to dynamically establish the pose of the camera.

たとえば、拡張現実では、入力画像をセグメント化して、カメラのポーズを動的に確立するために使用されるシーン内の既知のマーカーを見つける必要があります。

A simple and fast adaptive thresholding technique is therefore an important tool.

したがって、シンプルで高速な適応しきい値設定手法は重要なツールです。

**Integral Images**

An integral image (also known as a summed-area table) is a tool that can be used whenever we have a function from pixels to real numbers f (x,y) (for instance, pixel intensity), and we wish to compute the sum of this function over a rectangular region of the image.

積分画像（合計面積テーブルとも呼ばれます）は、ピクセルから実数f（x、y）への関数（ピクセル強度など）がある場合に使用できるツールです。そして、画像の長方形の領域でこの関数の合計を計算したいと思います。

Examples of where integral images have been applied include texture mapping [Crow 1984], face detection in images [Viola and Jones 2004], and stereo correspondence [Veksler 2003].

積分画像が適用された例には、テクスチャマッピング[Crow 1984]、画像での顔検出[Viola and Jones 2004]、ステレオ対応[Veksler 2003]が含まれます。

Without an integral image, the sum can be computed in linear time per rectangle by calculating the value of the function for each pixel individually.

積分画像がない場合、各ピクセルの関数値を個別に計算することにより、長方形ごとの線形時間で合計を計算できます。

However, if we need to compute the sum over multiple overlapping rectangular windows, we can use an integral image and achieve a constant number of operations per rectangle with only a linear amount of preprocessing.

ただし、複数のオーバーラップする長方形のウィンドウで合計を計算する必要がある場合は、積分イメージを使用して、線形の前処理だけで長方形ごとに一定数の操作を実現できます。

To compute the integral image, we store at each location, I(x,y), the sum of all f (x,y) terms to the left and above the pixel (x,y).

積分画像を計算するために、各位置に、ピクセルの左と上のすべての項の合計を保存します。

This is accomplished in linear time using the following equation for each pixel (taking into account the border cases),

これは、ピクセルごとに次の式を使用して線形時間で達成されます（境界ケースを考慮に入れて）。

Figure 2 (left and center) illustrates the computation of an integral image.

図2（左と中央）は、積分画像の計算を示しています。

Once we have the integral image, the sum of the function for any rectangle with upper left corner (x1,y1), and lower right corner (x2,y2) can be computed in constant time using the following equation,

積分画像が得られたら、左上隅（x1、y1）と右下隅（x2、y2）を持つ長方形の関数の合計は、次の式を使用して一定時間で計算できます。

Figure 2 (right) illustrates that computing the sum of f (x,y) over the rectangle D using Equation 2 is equivalent to computing the sums over the rectangles (A+B+C+D)-(A+B)-(A+C)+A.

図2（右）は、長方形Dでf（x、y）の合計を計算することを示しています

式2を使用することは、長方形（A + B + C + D）-（A + B）-（A + C）+ Aの合計を計算することと同等です。

要約

積分画像は次の式で計算できる。

次式を使用して、長方形（A + B + C + D）-（A + B）-（A + C）+ Aの合計を計算する。

**The Technique**

Our adaptive thresholding technique is a simple extension of Wellner’s method [Wellner 1993].

適応型しきい値処理手法は、Wellnerの手法を単純に拡張したものです[Wellner 1993]。

The main idea in Wellner’s algorithm is that each pixel is compared to an average of the surrounding pixels.

Wellnerのアルゴリズムの主なアイデアは、各ピクセルを周囲のピクセルの平均と比較することです。

Specifically, an approximate moving average of the last s pixels seen is calculated while traversing the image.

具体的には、最後に表示されたsピクセルの近似移動平均が、画像を走査しながら計算されます。

If the value of the current pixel is t percent lower than the average then it is set to black, otherwise it is set to white.

現在のピクセルの値が平均よりもtパーセント低い場合は黒に設定され、そうでない場合は白に設定されます。

This method works because comparing a pixel to the average of nearby pixels will preserve hard contrast lines and ignore soft gradient changes.

ピクセルを近くのピクセルの平均と比較すると、ハードコントラストラインが保持され、ソフトグラデーションの変化が無視されるため、この方法が機能します。

The advantage of this method is that only a single pass through the image is required. Wellner uses 1/8th of the image width for the value of s and 15 for the value of t.

この方法の利点は、イメージを1回通過するだけで済むことです。 Wellnerは、sの値にイメージ幅の1/8を使用し、tの値に15を使用します。

However, a problem with this method is that it is dependent on the scanning order of the pixels.

ただし、この方法にはピクセルのスキャン順序に依存する問題があります。

In addition, the moving average is not a good representation of the surrounding pixels at each step because the neighbourhood samples are not evenly distributed in all directions.

さらに、移動平均は、各サンプルで周囲のピクセルを適切に表現するものではありません。これは、近傍サンプルがすべての方向に均等に分布していないためです。

By using the integral image (and sacrificing one additional iteration through the image), we present a solution that does not suffer from these problems.

積分画像を使用して（およびイメージをさらに1回繰り返して犠牲にする）、これらの問題に悩まされないソリューションを提示します。

Our technique is clean, straightforward, easy to code, and produces the same output

independently of how the image is processed.

私たちのテクニックは、簡潔でわかりやすく、コーディングが簡単で、同じ出力を生成します。画像の処理方法とは関係ありません。

Instead of computing a running average of the last s pixels seen, we compute the average of an s x s window of pixels centered around each pixel.

最後に見たsピクセルの移動平均を計算する代わりに、各ピクセルを中心としたピクセルのs x sウィンドウの平均を計算します。

This is a better average for comparison since it considers neighbouring pixels on all sides.

これは、すべての辺の隣接ピクセルを考慮するため、比較のためのより良い平均です。

The average computation is accomplished in linear time by using the integral image.

平均計算は、積分画像を使用して線形時間で実行されます。

We calculate the integral image in the first pass through the input image.

入力画像の最初のパスで積分画像を計算します。

In a second pass, we compute the s x s average using the integral image for each pixel in constant time and then perform the comparison.

2番目のパスでは、一定時間で各ピクセルの積分画像を使用してs x s平均を計算し、比較を実行します。

If the value of the current pixel is t percent less than this average then it is set to black, otherwise it is set to white.

現在のピクセルの値がこの平均よりもtパーセント小さい場合は黒に設定され、そうでない場合は白に設定されます。

The following pseudocode demonstrates our technique for input image in, output binary image out, image width w and image height h.

次の擬似コードは、入力画像(in)、出力バイナリ画像(out)、画像幅(w)および画像高さ(h)の手法を示しています。

要約

**Performance and Examples**

We tested our real-time adaptive thresholding technique on a Pentium 4 processor at 3.4Ghz with a Point Grey Dragonfly camera, capturing at a resolution of 640 x 480 pixels.

640 x 480ピクセルの解像度でキャプチャするPoint Grey Dragonflyカメラを使用して、3.4 GhzのPentium 4プロセッサでリアルタイムの適応しきい値処理技術をテストしました。

On average our technique operates at approximately 15 milliseconds per frame, yielding a frame-rate of 65 frames per second.

平均して、この手法は1フレームあたり約15ミリ秒で動作し、1秒あたり65フレームのフレームレートを生成します。

Since we require two iterations through each image rather than just one as in Wellner’s method, we expect to perform slower.

Wellnerの方法のように1回だけではなく、各画像を2回繰り返す必要があるため、パフォーマンスが低下することが予想されます。

In practice we measured our technique at approximately 2.5 times slower than that of Wellner, however we still achieve a real-time frame-rate and a better segmentation.

実際には、Wellnerの約2.5倍遅い速度でテクニックを測定しましたが、リアルタイムフレームレートとより良いセグメンテーションを達成しています。

Discussion

Our previous work in augmented reality [Bradley and Roth 2004] demonstrates a need for robust adaptive thresholding of real-time video streams.

拡張現実における以前の研究[Bradley and Roth 2004]は、リアルタイムビデオストリームの堅牢な適応しきい値の必要性を示しています。

Our technique is well-suited for scenes with strong spatial changes in illumination.

この手法は、照明の空間的変化が激しいシーンに適しています。

Temporal variations in illumination are also handled automatically, which is not the case for global thresholding methods.

照明の時間的変動も自動的に処理されますが、これはグローバルなしきい値設定方法には当てはまりません。

The main drawback of our method is that we must process the image twice.

この方法の主な欠点は、画像を2回処理する必要があることです。

However, this is not a significant drawback with the current speed of processors and our technique is still suitable for real-time applications.

ただし、これは現在のプロセッサの速度にとって重大な欠点ではなく、この手法はリアルタイムアプリケーションに適しています。

The amount of processing can also be minimized by performing the thresholding step at pixel (i−s/2, j−s/2) at the same time as computing the integral image at pixel (i, j).

ピクセル（i、j）で積分画像を計算すると同時に、ピクセル（i-s / 2、j-s / 2）でしきい値設定ステップを実行することにより、処理量を最小化することもできます。

In this case, a constant amount of processing must be performed for the following number of pixels,

この場合、次のピクセル数に対して一定量の処理を実行する必要があります。

This technique for thresholding makes the assumption that the image contains primarily background pixels (to be segmented as white) and that the foreground pixels are rather distributed in the image.

このしきい値処理の手法では、画像に主に背景ピクセル（白としてセグメント化される）が含まれ、前景ピクセルが画像内にかなり分布していると仮定します。

In the case of a foreground component that is larger than s x s pixels, the center of the component will be incorrectly classified as background.

s x sピクセルよりも大きい前景コンポーネントの場合、コンポーネントの中心は誤って背景として分類されます。

This problem can be alleviated by using a larger kernel (ie: by increasing s), however fine details in the segmentation may then be lost.

この問題は、より大きなカーネルを使用することで軽減できます（つまり、sを増やすことで）。ただし、セグメンテーションの詳細が失われる可能性があります。

The kernel size should be set appropriately according to the application.

カーネルサイズは、アプリケーションに応じて適切に設定する必要があります。