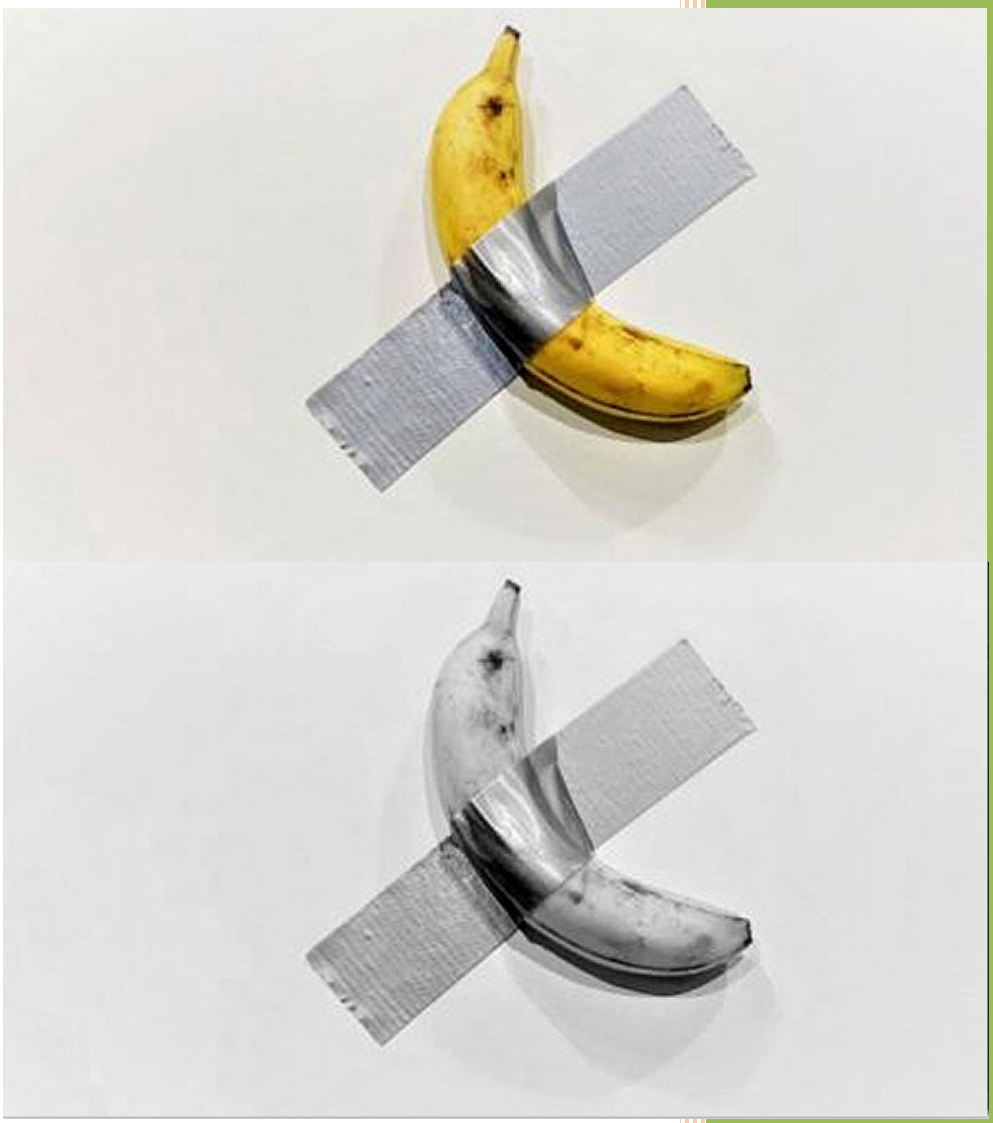
2020

Meetrapport Efficiëntie RGB Conversie



Jeffrey de Waal & Diego Nijboer

Vision, Diederik Yamamoto-Roijers

4/4/2020

Inhoudsopgave

1.1. Doel

Ons doel van dit meetrapport is het uitzoeken wat de efficiëntste manier is om van afbeeldingen de RGB waardes om te zetten naar grijswaarden. De definitie van efficiëntie hierbij is de snelheid & precisie bij het omzetten van de RGB waardes naar grijswaarden.

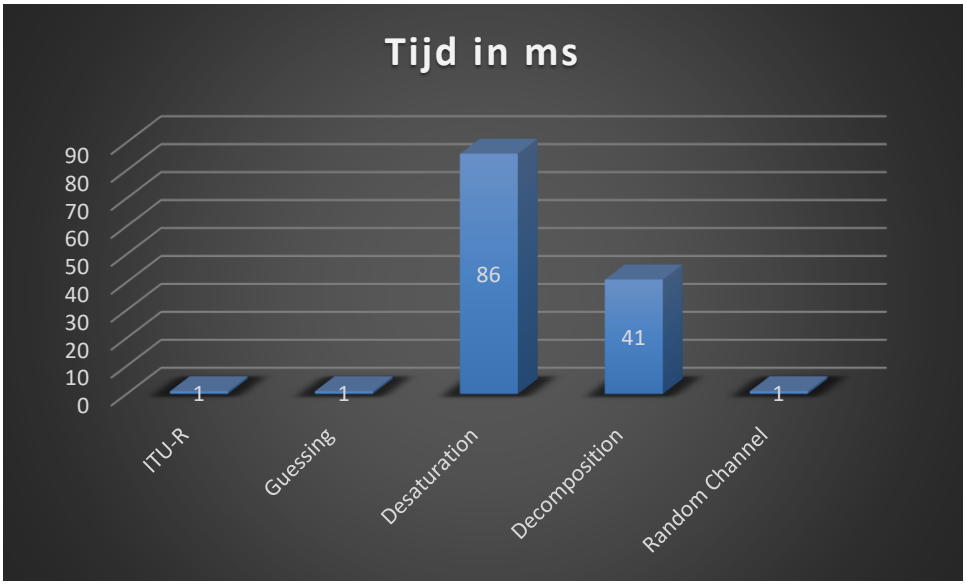
1.2. Hypothese

Wij verwachten dat de 'Guessing' conversie methode het beste zal presteren omdat deze in vergelijking met de andere conversie methoden relatief weinig rekenkracht nodig heeft.

1.3. Werkwijze

Doormiddel van code waarmee we de snelheid van alle methodes kunnen meten, kunnen we per methode de snelheid waarmee 150,000 RGB objecten omgezet worden naar intensity objecten bepalen. De code hiervan is aan het einde van dit document bijgevoegd.

1.4. Resultaten



Conversie Methode	Tijd in ms
ITU-R	1
Guessing	1
Desaturation	86
Decomposition	41
Random Channel	1

Tot nu toe zien wij dat qua snelheid ITU-R, Guessing & Random Channel het efficiëntst zijn, op basis van alleen deze resultaten kunnen we nog geen conclusie trekken, dus besloten we om ook te kijken naar de precisie van de verschillende methoden.

Het origineel:

ITU-R:

Guessing:

Desaturation:

Decomposition:

Min

Max

Random Channel:

R

G

B

Aan de details vallen ons een aantal dingen op:

Conversie Methode	Details
ITU-R	Deze ziet er van alle resultaten het meest scherp uit.
Guessing	Erg vergelijkbaar met ITU-R, alleen toch net wat minder qua kwaliteit.
Desaturation	Ziet er net wat meer blurry uit dan Guessing en ITU-R.
Decomposition	De details en edges zijn wat minder scherp.
Random Channel	De details en edges zijn het minst scherp.

1.5. Verwerking

Ten eerste keken we naar de snelheid van de verschillende conversie methoden, toen dat geen doorslag gaf in welke het meest efficiënt is, hebben we ook naar de resultaten van de images gekeken om te bepalen wat de resultaten van de methoden waren. Aan de hand daarvan hebben we naar de details van elke image gekeken.

1.6. Conclusie

ITU-R levert de beste resultaten op en is gelijkwaardig qua snelheid met de andere snelste conversie methoden. Dat maakt ITU-R de efficiëntste methode uit ons onderzoek, aangezien het snelheidsverschil verwaarloosbaar en het de beste resultaten oplevert wat betreft edges en details.

1.7. Evaluatie

Onze verwachting dat Guessing de beste methode zou zijn kwam in de buurt aangezien is het verschil tussen ITU-R en Guessing vrij klein is. Toch zouden we zeggen dat ITU-R de efficiëntere methode is vanwege het verschil in precisie van edges.

We hebben rekening gehouden met meetfouten door een hoog aantal objecten waar de algoritmes op draaide om te zetten, hierdoor werd de tijd voor elk algoritme groter, waardoor we vervolgens duidelijker het verschil in tijd konden zien. De conclusie over details waren deels subjectief, maar we hebben geprobeerd om zoveel mogelijk naar de ‘harde verschillen’ in edges te kijken, zodat onze mening gebaseerd is op feiten.

1.8. Code voor de snelheidstest

Voor het tonen van de afbeeldingen:

```
float parse_rgb::execute_guess()
{
    std::cout << "Guessing: ";
    begin_time = clock();
    for (int i = 0; i < 150000; i++) {
        intensity_container[i].value = (rgb_container[i].r + rgb_container[i].g + rgb_container[i].b) / 3;
    }
    return float(clock() - begin_time) / CLOCKS_PER_SEC;
}
float parse_rgb::execute_itu_r()
{
    std::cout << "ITU-R: ";
    begin_time = clock();
    for (int i = 0; i < 150000; i++) {
        intensity_container[i].value = (int)(0.2126 * rgb_container[i].r + 0.7152 * rgb_container[i].g + 0.0722 * rgb_container[i].b);
    }
    return float(clock() - begin_time) / CLOCKS_PER_SEC;
}
float parse_rgb::execute_desaturation()
{
    std::cout << "Desaturation: ";
    begin_time = clock();
    for (int i = 0; i < 150000; i++) {
        intensity_container[i].value = (.(std::max(std::max((int)rgb_container[i].r, (int)rgb_container[i].g), (int)rgb_container[i].b) +
        (std::min(std::max((int)rgb_container[i].r, (int)rgb_container[i].g), / / (int)rgb_container[i].b)))) / 2;
    }
    return float(clock() - begin_time) / CLOCKS_PER_SEC;
}
float parse_rgb::execute_decomposition()
{
    std::cout << "Decomposition: ";
    begin_time = clock();
    for (int i = 0; i < 150000; i++) {
        intensity_container[i].value = std::max(std::max((int)rgb_container[i].r, (int)rgb_container[i].g), (int)rgb_container[i].b);
    }
    return float(clock() - begin_time) / CLOCKS_PER_SEC;
}
float parse_rgb::execute_random_channel()
{
    std::cout << "Random channel: ";
    begin_time = clock();
    for (int i = 0; i < 150000; i++) {
        intensity_container[i].value = rgb_container[i].r;
    }
    return float(clock() - begin_time) / CLOCKS_PER_SEC;
}
```

Voor de snelheidstest:

```
Intensity IntensityImageStudent::guessing(RGB& value)
{
    Intensity new_value = (value.r + value.g + value.b) / 3;
    return new_value;
}
Intensity IntensityImageStudent::itu_r(RGB& value)
{
    Intensity new_value = (int)(0.2126 * value.r + 0.7152 * value.g + 0.0722 * value.b);
    return new_value;
}
Intensity IntensityImageStudent::desaturation(RGB& value)
{
    int list[3] = { (int)(value.r), (int)(value.g), (int)(value.b) };
    Intensity new_value = ((std::max(std::max((int)list[0], (int)list[1]), (int)list[2]) +
    (std::min(std::max((int)list[0], (int)list[1]), (int)list[2])))) / 2;
    return new_value;
}
Intensity IntensityImageStudent::decomposition(RGB& value)
{
    int list[3] = { (int)(value.r), (int)(value.g), (int)(value.b) };
    Intensity new_value = std::max(std::max((int)list[0], (int)list[1]), (int)list[2]);
    Intensity new_value = std::min(std::max((int)list[0], (int)list[1]), (int)list[2]);
    return new_value;
}
Intensity IntensityImageStudent::random(RGB& value)
{
    Intensity new_value = value.r;
    Intensity new_value = value.g;
    Intensity new_value = value.b;
    return new_value;
}
```