Making 3D Binary Digital Images Well-Composed

Marcelo Siqueira
University of Pennsylvania, USA
marcelos@seas.upenn.edu

Longin Jan Latecki
Temple University, USA
latecki@temple.edu

Jean Gallier
University of Pennsylvania, USA
jean@cis.upenn.edu

Introduction

In this talk we present

Introduction

In this talk we present

- a **new algorithm** to make 3D binary digital images well-composed,

Introduction

In this talk we present

- a **new algorithm** to make 3D binary digital images well-composed,
- a **probabilistic bound** for the (theoretical) effectiveness of our algorithm, and

Introduction

In this talk we present

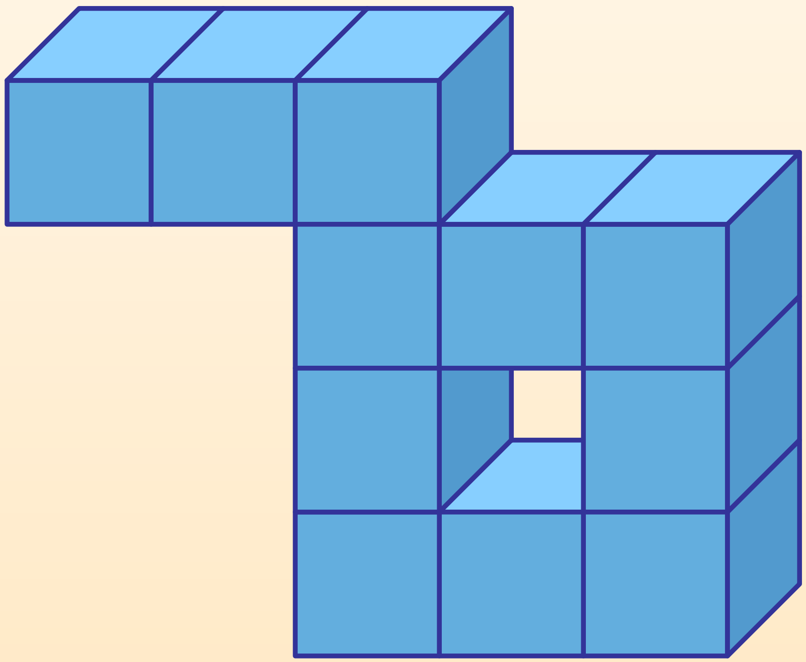
- a **new algorithm** to make 3D binary digital images well-composed,
- a **probabilistic bound** for the (theoretical) effectiveness of our algorithm, and
- several **examples** of the application of our algorithm to magnetic resonance (MR) images.

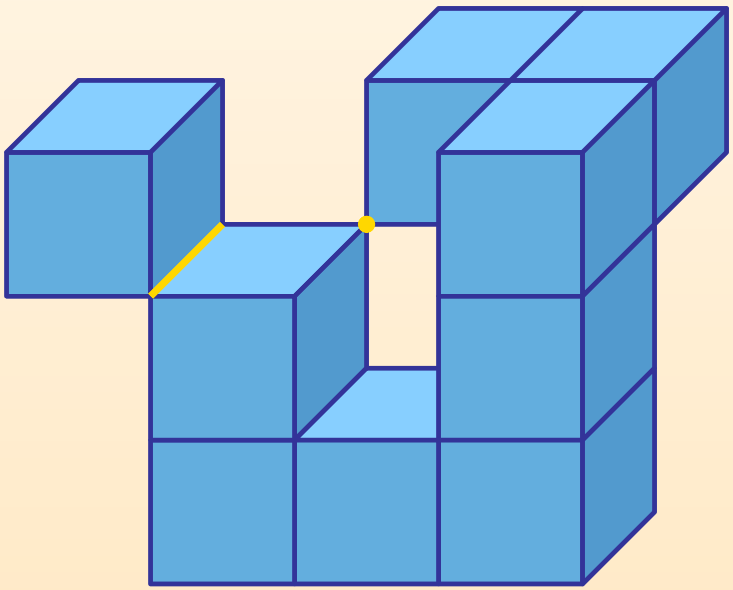
Introduction

A 3D binary digital image is said to be **well-composed** if and only if the set of points in the voxel faces shared by foreground and background points of the image is a 2D manifold.

Introduction

A 3D binary digital image is said to be **well-composed** if and only if the set of points in the voxel faces shared by foreground and background points of the image is a 2D manifold.





Introduction

Why is well-composed a desirable property?

Introduction

Why is well-composed a desirable property?

- Well-composed images enjoy nice topological and geometric properties, such as the fact that there is only one connectedness relation on points of the image, i.e., 6-, 18-, 26-connected components are equal.

Introduction

Why is well-composed a desirable property?

- Well-composed images enjoy nice topological and geometric properties, such as the fact that there is only one connectedness relation on points of the image, i.e., 6-, 18-, 26-connected components are equal.

- Several algorithms that deal with binary images become simpler if the input image is well-composed (e.g., thinning algorithms and grid-based algorithms for extracting iso-surfaces).

Introduction

Why is well-composed a desirable property?

- Well-composed images enjoy nice topological and geometric properties, such as the fact that there is only one connectedness relation on points of the image, i.e., 6-, 18-, 26-connected components are equal.

- Several algorithms that deal with binary images become simpler if the input image is well-composed (e.g., thinning algorithms and grid-based algorithms for extracting iso-surfaces).

What if a 3D binary digital image is not well-composed?

Introduction

Why is well-composed a desirable property?

- Well-composed images enjoy nice topological and geometric properties, such as the fact that there is only one connectedness relation on points of the image, i.e., 6-, 18-, 26-connected components are equal.

- Several algorithms that deal with binary images become simpler if the input image is well-composed (e.g., thinning algorithms and grid-based algorithms for extracting iso-surfaces).

What if a 3D binary digital image is not well-composed?

- A previous result shows that the **digitization process** that gave rise to the (ill-composed) image is not **topology-preserving.** Otherwise, the image would be well-composed!

Introduction

The previous remark motivated the development of **repairing algorithms**, i.e., algorithms for restoring the well-composedness property of ill-composed images.

Introduction

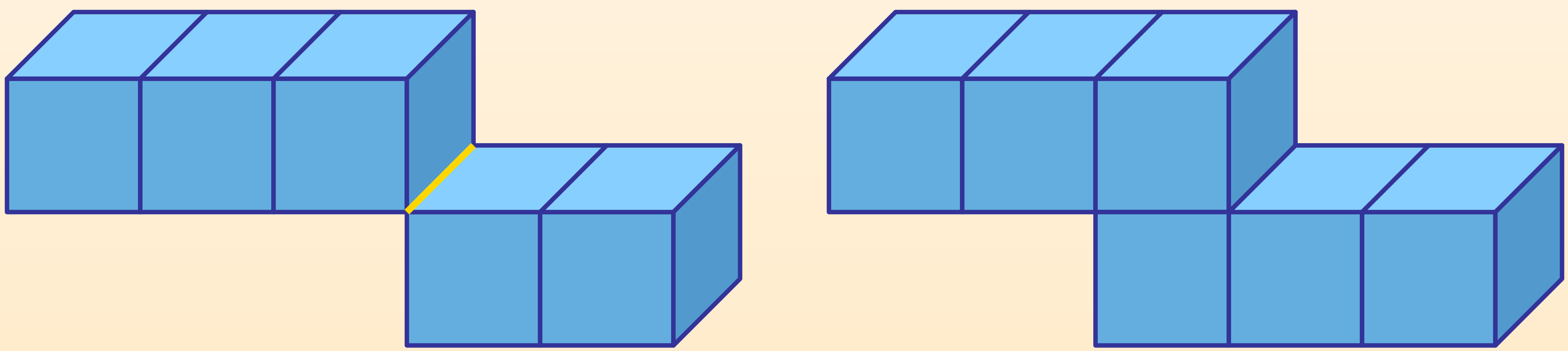
The previous remark motivated the development of **repairing algorithms**, i.e., algorithms for restoring the well-composedness property of ill-composed images.

The idea behind a repairing algorithm is to **change the value (i.e., color)** assigned with some voxels of the ill-composed image such that the well-composedness property is restored.

Introduction

The previous remark motivated the development of **repairing algorithms**, i.e., algorithms for restoring the well-composedness property of ill-composed images.

The idea behind a repairing algorithm is to **change the value (i.e., color)** assigned with some voxels of the ill-composed image such that the well-composedness property is restored.



Before After

Introduction

In general, repairing algorithms cannot guarantee that the resulting well-composed image is the one that would have been obtained if a topology-preserving digitization process had been used.

Introduction

In general, repairing algorithms cannot guarantee that the resulting well-composed image is the one that would have been obtained if a topology-preserving digitization process had been used.

However, if the resulting well-composed image is ‘‘similar’’ to the ill-composed one, a repairing algorithm can be very useful in the context of several image-based applications.

Introduction

In general, repairing algorithms cannot guarantee that the resulting well-composed image is the one that would have been obtained if a topology-preserving digitization process had been used.

However, if the resulting well-composed image is ‘‘similar’’ to the ill-composed one, a repairing algorithm can be very useful in the context of several image-based applications.

What do we mean by ‘‘similar’’? How do we measure similarity?

Introduction

In general, repairing algorithms cannot guarantee that the resulting well-composed image is the one that would have been obtained if a topology-preserving digitization process had been used.

However, if the resulting well-composed image is ‘‘similar’’ to the ill-composed one, a repairing algorithm can be very useful in the context of several image-based applications.

What do we mean by ‘‘similar’’? How do we measure similarity?

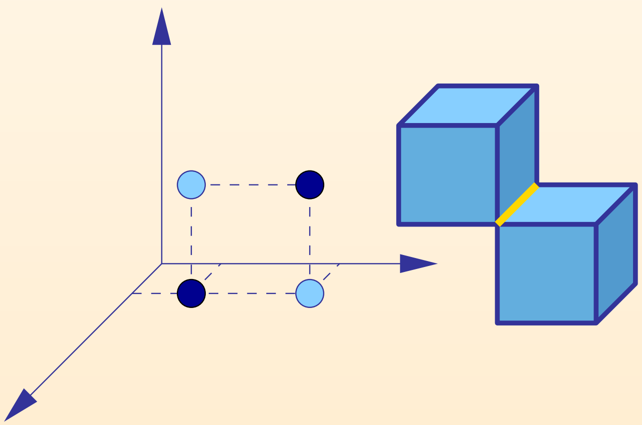
We now describe our algorithm for repairing ill-composed images.

The Repairing Algorithm

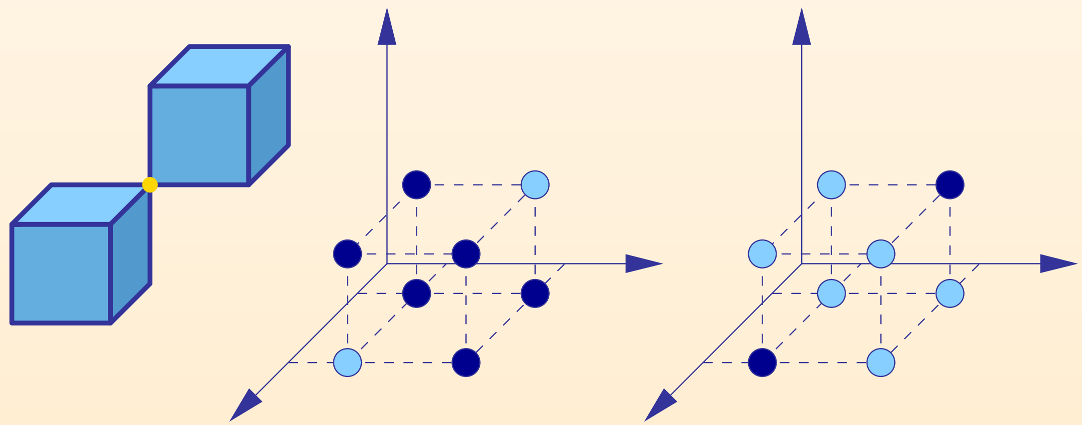
Our repairing algorithm makes use of the fact (proved elsewhere) that a 3D digital binary image has the well-composedness property if and only if the image does not contain any instance of the following two **critical configurations** (up to reflections and rotations):

The Repairing Algorithm

Our repairing algorithm makes use of the fact (proved elsewhere) that a 3D digital binary image has the well-composedness property if and only if the image does not contain any instance of the following two **critical configurations** (up to reflections and rotations):



Critical Configuration 1 (C1)



Critical Configuration 2 (C2)

The Repairing Algorithm

The strategy used by our algorithm is to initially let the output image be the same as the input image, and then remove all instances of (C1) and (C2) from the output image.

The Repairing Algorithm

The strategy used by our algorithm is to initially let the output image be the same as the input image, and then remove all instances of (C1) and (C2) from the output image.

Instances of critical configurations are removed by converting background points into foreground points, i.e., by **changing the color of background points**.

The Repairing Algorithm

The strategy used by our algorithm is to initially let the output image be the same as the input image, and then remove all instances of (C1) and (C2) from the output image.

Instances of critical configurations are removed by converting background points into foreground points, i.e., by **changing the color of background points**.

There are two main issues related to this strategy:

The Repairing Algorithm

The strategy used by our algorithm is to initially let the output image be the same as the input image, and then remove all instances of (C1) and (C2) from the output image.

Instances of critical configurations are removed by converting background points into foreground points, i.e., by **changing the color of background points.**

There are two main issues related to this strategy:

- Can our repairing algorithm always generate a well-composed image?

The Repairing Algorithm

The strategy used by our algorithm is to initially let the output image be the same as the input image, and then remove all instances of (C1) and (C2) from the output image.

Instances of critical configurations are removed by converting background points into foreground points, i.e., by **changing the color of background points.**

There are two main issues related to this strategy:

- Can our repairing algorithm always generate a well-composed image?
- If so, which background points should be converted into foreground points so that all critical configurations are removed? And how similar are the input and the output images?

The Repairing Algorithm

A well-composed image can always be generated in this way, as the conversion of **all background points** into foreground points is guaranteed to generate a well-composed image.

The Repairing Algorithm

A well-composed image can always be generated in this way, as the conversion of **all background points** into foreground points is guaranteed to generate a well-composed image.

So, the question is really to find **a smallest subset of background points** such that the conversion into foreground points yields a well-composed image.

The Repairing Algorithm

A well-composed image can always be generated in this way, as the conversion of **all background points** into foreground points is guaranteed to generate a well-composed image.

So, the question is really to find **a smallest subset of background points** such that the conversion into foreground points yields a well-composed image.

We can easily find such a smallest set by enumerating all possible subsets of background points in decreasing order of cardinality.

The Repairing Algorithm

A well-composed image can always be generated in this way, as the conversion of **all background points** into foreground points is guaranteed to generate a well-composed image.

So, the question is really to find **a smallest subset of background points** such that the conversion into foreground points yields a well-composed image.

We can easily find such a smallest set by enumerating all possible subsets of background points in decreasing order of cardinality.

The problem with the above approach is that its performance can be really poor, as there are 2^n subsets of background points in an image with n background points.

The Repairing Algorithm

So, we propose an alternative approach that computes a **possibly small** set of background points in linear time in the number of points of the image.

The Repairing Algorithm

So, we propose an alternative approach that computes a **possibly small** set of background points in linear time in the number of points of the image.

The key idea behind our approach is to exploit the ‘‘locality’’ of critical configurations.

The Repairing Algorithm

So, we propose an alternative approach that computes a **possibly small** set of background points in linear time in the number of points of the image.

The key idea behind our approach is to exploit the ‘‘locality’’ of critical configurations.

Let (D, X) denote a 3D binary digital image, where D is the set of points of the image (i.e, the image domain), and $X \subseteq D$ is the set of foreground points of the image.

The Repairing Algorithm

So, we propose an alternative approach that computes a **possibly small** set of background points in linear time in the number of points of the image.

The key idea behind our approach is to exploit the ''locality'' of critical configurations.

Let (D, X) denote a 3D binary digital image, where D is the set of points of the image (i.e, the image domain), and $X \subseteq D$ is the set of foreground points of the image.

Let $P \subseteq (D \setminus X)$ denote the set of background points obtained by our algorithm, where $D \setminus X$ is the complement set of X with respect to D, i.e., the set of background points of (D, X).

The Repairing Algorithm

Our algorithm computes P iteratively.

The Repairing Algorithm

Our algorithm computes P iteratively.

Let P^i denote the set P at iteration i.

The Repairing Algorithm

Our algorithm computes P iteratively.

Let P^i denote the set P at iteration i.

Let $(D, X \cup P^i)$ denote the 3D binary digital image obtained from (D, X) by converting the set P^i of background points of (D, X) into foreground points.

The Repairing Algorithm

Our algorithm computes P iteratively.

Let P^i denote the set P at iteration i.

Let $(D, X \cup P^i)$ denote the 3D binary digital image obtained from (D, X) by converting the set P^i of background points of (D, X) into foreground points.

At each iteration i of the algorithm, one or more points from $D \setminus P^i$ is inserted into P^i.

The Repairing Algorithm

Our algorithm computes P iteratively.

Let P^i denote the set P at iteration i.

Let $(D, X \cup P^i)$ denote the 3D binary digital image obtained from (D, X) by converting the set P^i of background points of (D, X) into foreground points.

At each iteration i of the algorithm, one or more points from $D \setminus P^i$ is inserted into P^i.

The insertion of a point from $D \setminus P^i$ into P^i eliminates at least one instance of a critical configuration of $(D, X \cup P^i)$.

The Repairing Algorithm

In the following, we give a pseudocode for our algorithm:

The Repairing Algorithm

In the following, we give a pseudocode for our algorithm:

- Let $i = 1$ and let $P = \emptyset$.
- While $(D, X \cup P^i)$ is not well-composed do
  - ▷ *Find all instances of (C1) and (C2) in* $(D, X \cup P^i)$ *and eliminate them.*
  - ▷ *Increment* i *by* 1
- Output $(D, X \cup P)$.

The Repairing Algorithm

In the following, we give a pseudocode for our algorithm:

- Let $i = 1$ and let $P = \emptyset$.
- While $(D, X \cup P^i)$ is not well-composed do
  - ▷ *Find all instances of (C1) and (C2) in* $(D, X \cup P^i)$ *and eliminate them.*
  - ▷ *Increment* i *by* 1
- Output $(D, X \cup P)$.

Note that, for $i \geq 2$, if $(D, X \cup P^i)$ is not well-composed, then all instances of (C1) and (C2) in $(D, X \cup P^i)$ were created during the elimination of all instances of (C1) and (C2) in $(D, X \cup P^{i-1})$.

The Repairing Algorithm

To find all instances of (C1) and (C2) in $(D, X \cup P^1) = (D, X \cup P)$, we scan the entire set D and examine every 2×2 and $2 \times 2 \times 2$ subsets of points of D.

The Repairing Algorithm

To find all instances of (C1) and (C2) in $(D, X \cup P^1) = (D, X \cup P)$, we scan the entire set D and examine every 2×2 and $2 \times 2 \times 2$ subsets of points of D.

For $i \geq 2$, we do not scan D again. Instead, we keep track of all new critical configurations created during the elimination of all instances of (C1) and (C2) in $(D, X \cup P^{i-1})$, as they are the only critical configurations of $(D, X \cup P^i)$.

The Repairing Algorithm

To find all instances of (C1) and (C2) in $(D, X \cup P^1) = (D, X \cup P)$, we scan the entire set D and examine every 2×2 and $2 \times 2 \times 2$ subsets of points of D.

For $i \geq 2$, we do not scan D again. Instead, we keep track of all new critical configurations created during the elimination of all instances of (C1) and (C2) in $(D, X \cup P^{i-1})$, as they are the only critical configurations of $(D, X \cup P^i)$.

Each critical configuration of $(D, X \cup P^i)$ is eliminated by the insertion of **exactly one** background point from $D \setminus P^i$ into P^i.

The Repairing Algorithm

To find all instances of (C1) and (C2) in $(D, X \cup P^1) = (D, X \cup P)$, we scan the entire set D and examine every 2×2 and $2 \times 2 \times 2$ subsets of points of D.

For $i \geq 2$, we do not scan D again. Instead, we keep track of all new critical configurations created during the elimination of all instances of (C1) and (C2) in $(D, X \cup P^{i-1})$, as they are the only critical configurations of $(D, X \cup P^i)$.

Each critical configuration of $(D, X \cup P^i)$ is eliminated by the insertion of **exactly one** background point from $D \setminus P^i$ into P^i.

The choice of a background point from $D \setminus P^i$ to insert into P^i is the key step of our algorithm.

The Repairing Algorithm

How do we choose a point from $D \setminus P^i$ to insert into P^i?

The Repairing Algorithm

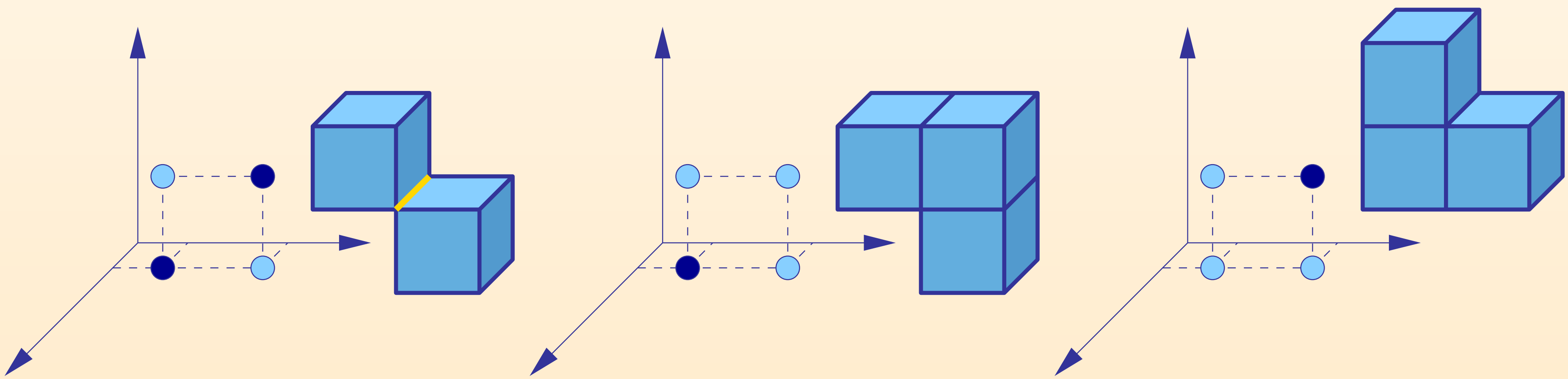
How do we choose a point from $D \setminus P^i$ to insert into P^i?

By observing the picture below, we see that a critical configuration 1 can **only be removed if** one (or both) of its two background points is made into a foreground point.

The Repairing Algorithm

How do we choose a point from $D \setminus P^i$ to insert into P^i?

By observing the picture below, we see that a critical configuration 1 can **only be removed if** one (or both) of its two background points is made into a foreground point.



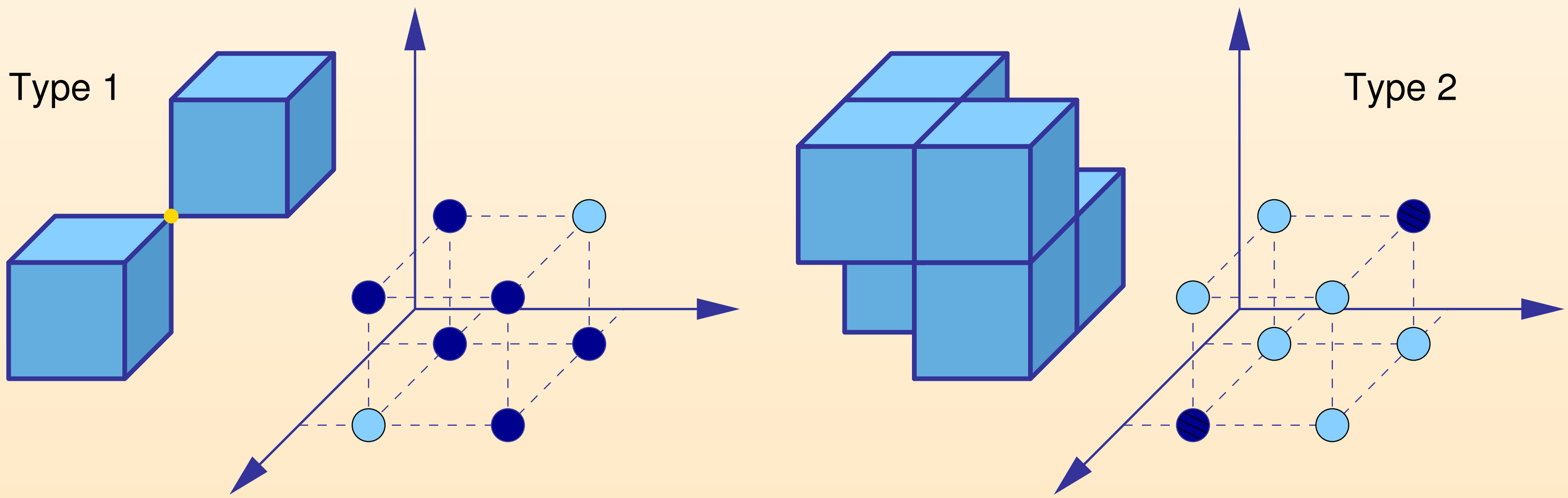
The Repairing Algorithm

Unlike instances of (C1), an instance of (C2) can be of one of two types:

The Repairing Algorithm

Unlike instances of (C1), an instance of (C2) can be of one of two types:

- Type 1: There are **six background** points and **two foreground** points in the instance of (C2).
- Type 2: There are **two background** points and **six foreground** points in the instance of (C2).

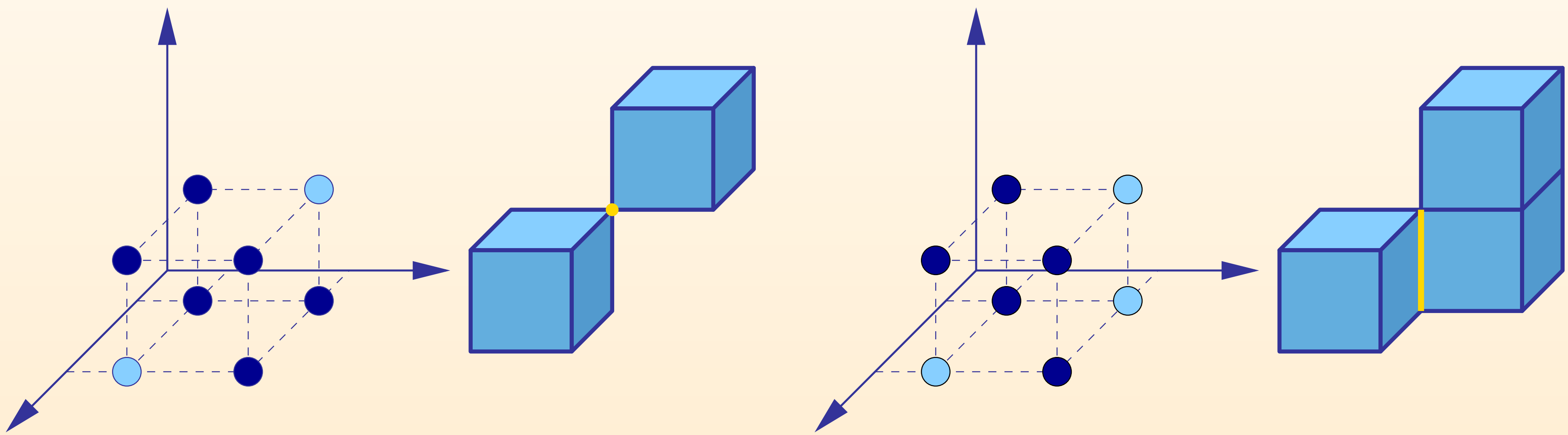


The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 1 can **only be removed if** one (or more) of its six background points is made into a foreground point.

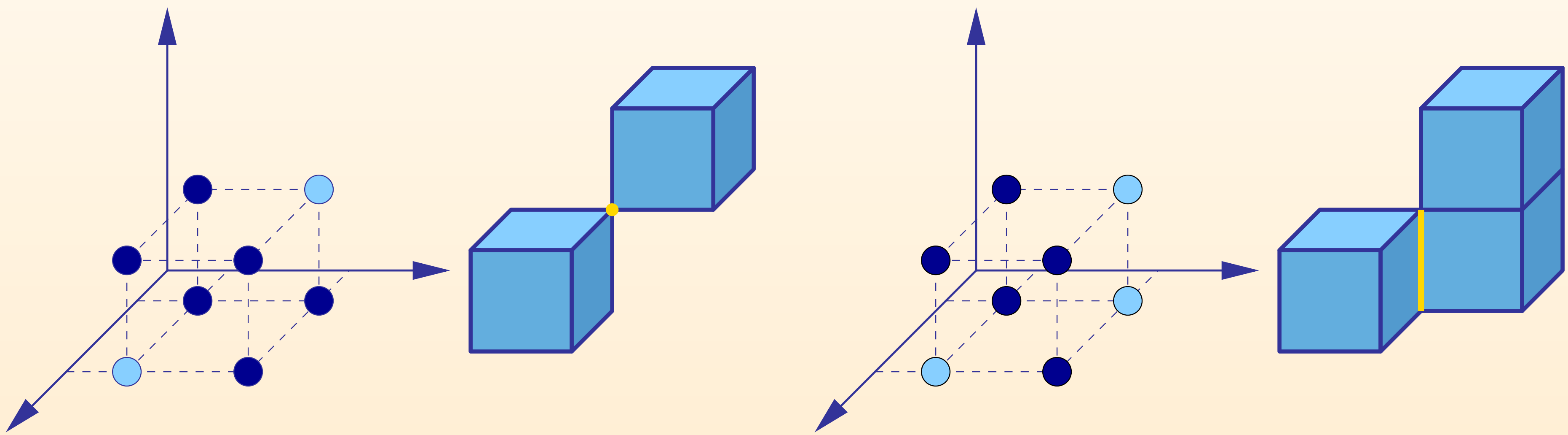
The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 1 can **only be removed if** one (or more) of its six background points is made into a foreground point.



The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 1 can **only be removed if** one (or more) of its six background points is made into a foreground point.



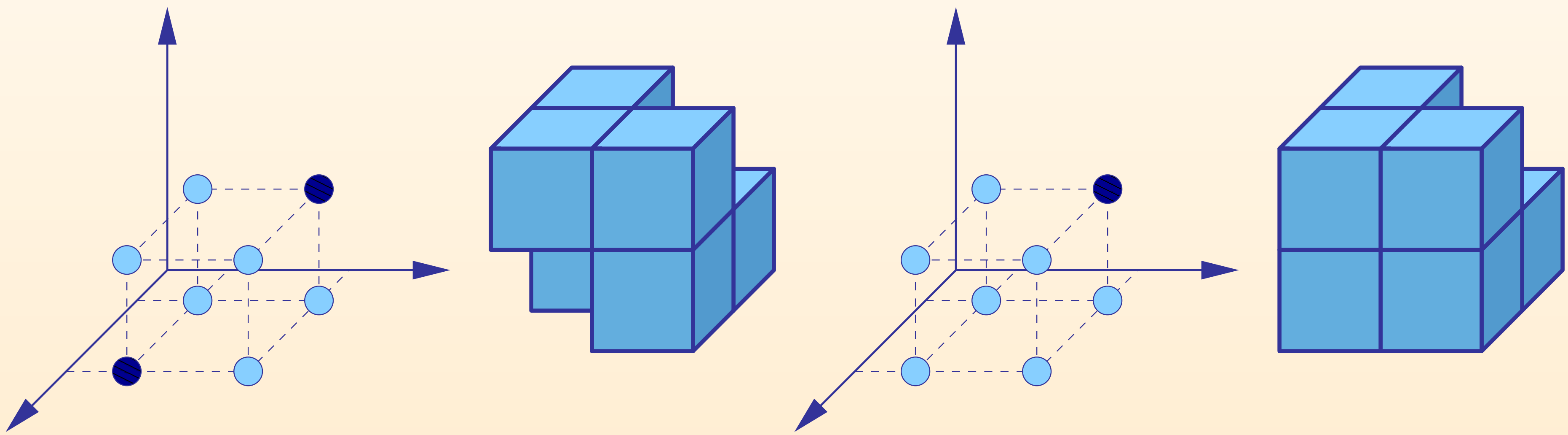
However, the conversion of one background point into a foreground point **always** gives rise to **a new** critical configuration 1.

The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 2 can **only be removed if** one (or both) of its two background points is made into a foreground point.

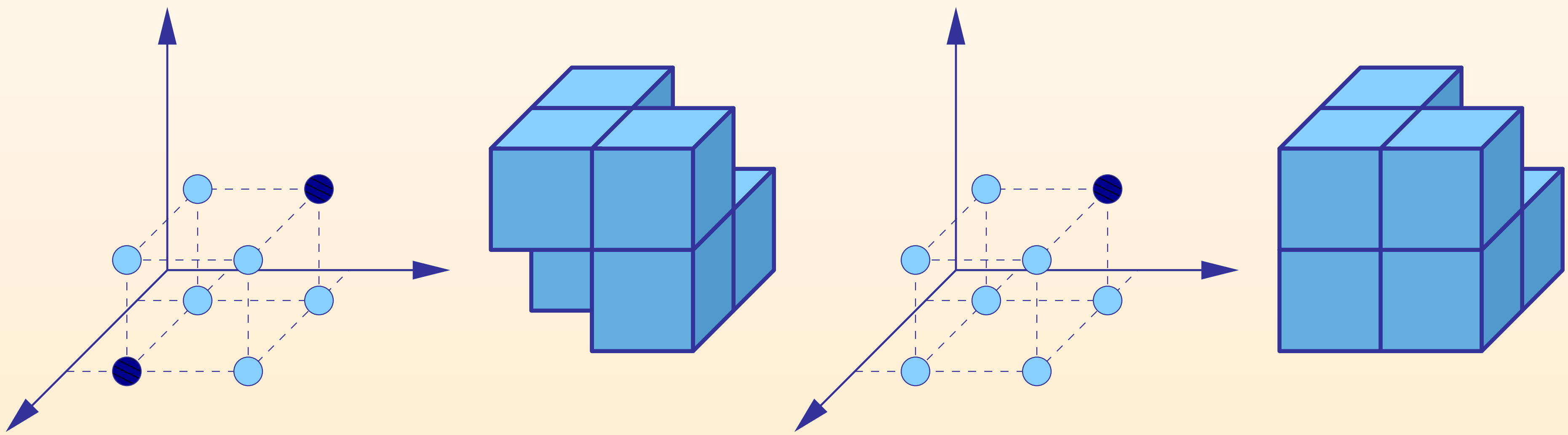
The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 2 can **only be removed if** one (or both) of its two background points is made into a foreground point.



The Repairing Algorithm

By observing the picture below, we see that a critical configuration 2 of type 2 can **only be removed if** one (or both) of its two background points is made into a foreground point.



This case is similar to the case of removal of an instance of (C1).

The Repairing Algorithm

Let $CC \subset D$ be an instance of a critical configuration (C1) or (C2), and let S be the set of background points of CC.

The Repairing Algorithm

Let $CC \subset D$ be an instance of a critical configuration (C1) or (C2), and let S be the set of background points of CC.

If CC is an instance of $(C1)$ or an instance of $(C2)$ of type 2, then S has exactly two background points of $D \setminus P^i$. Otherwise, CC is an instance of $(C2)$ of type 1 and therefore S contains exactly six background points of $D \setminus P^i$.

The Repairing Algorithm

Let $CC \subset D$ be an instance of a critical configuration (C1) or (C2), and let S be the set of background points of CC.

If CC is an instance of $(C1)$ or an instance of $(C2)$ of type 2, then S has exactly two background points of $D \setminus P^i$. Otherwise, CC is an instance of $(C2)$ of type 1 and therefore S contains exactly six background points of $D \setminus P^i$.

We know that the conversion of any point of S into a foreground point eliminates CC. However, such a conversion may or may not create a new configuration.

The Repairing Algorithm

Let $CC \subset D$ be an instance of a critical configuration (C1) or (C2), and let S be the set of background points of CC.

If CC is an instance of $(C1)$ or an instance of $(C2)$ of type 2, then S has exactly two background points of $D \setminus P^i$. Otherwise, CC is an instance of $(C2)$ of type 1 and therefore S contains exactly six background points of $D \setminus P^i$.

We know that the conversion of any point of S into a foreground point eliminates CC. However, such a conversion may or may not create a new configuration.

Let $S' \subseteq S$ be the subset of points q of S that can be converted into foreground points without giving rise to a new critical configuration in $(D, X \cup (P^i \cup q))$.

The Repairing Algorithm

Our repairing algorithm computes S' and applies the following three **mutually exclusive** rules:

The Repairing Algorithm

Our repairing algorithm computes S' and applies the following three **mutually exclusive** rules:

- If S' contains exactly one element, then choose this element to insert into P^i.

The Repairing Algorithm

Our repairing algorithm computes S' and applies the following three **mutually exclusive** rules:

- If S' contains exactly one element, then choose this element to insert into P^i.
- If S' contains more than one element, then **randomly** choose any point from S' to insert into P^i.

The Repairing Algorithm

Our repairing algorithm computes S' and applies the following three **mutually exclusive** rules:

- If S' contains exactly one element, then choose this element to insert into P^i.
- If S' contains more than one element, then **randomly** choose any point from S' to insert into P^i.
- If S' is empty, then **randomly** choose any point from S to insert into P^i and keep track of all critical configurations created (in $(D, X \cup P^{i+1})$) by the conversion of this point into a foreground one.

The Repairing Algorithm

Our repairing algorithm computes S' and applies the following three **mutually exclusive** rules:

- If S' contains exactly one element, then choose this element to insert into P^i.
- If S' contains more than one element, then **randomly** choose any point from S' to insert into P^i.
- If S' is empty, then **randomly** choose any point from S to insert into P^i and keep track of all critical configurations created (in $(D, X \cup P^{i+1})$) by the conversion of this point into a foreground one.

By applying any of these rules, our algorithm is guaranteed to eliminate CC. Furthermore, a new critical configuration is created if and only if the third rule is applied. This new critical configuration will be eliminated in the next iteration.

The Repairing Algorithm

The following observations imply the termination of our algorithm:

The Repairing Algorithm

The following observations imply the termination of our algorithm:

- There are finitely many background points in (D, X), and there are also finitely many critical configurations in (D, X).

The Repairing Algorithm

The following observations imply the termination of our algorithm:

- There are finitely many background points in (D, X), and there are also finitely many critical configurations in (D, X).
- Only background points (from $D \setminus P^i$) are inserted into P^i, and no point is removed from P^i.

The Repairing Algorithm

The following observations imply the termination of our algorithm:

- There are finitely many background points in (D, X), and there are also finitely many critical configurations in (D, X).
- Only background points (from $D \setminus P^i$) are inserted into P^i, and no point is removed from P^i.
- Every iteration i of our algorithm inserts at least one point into P^i, and each point insertion eliminates at least one critical configuration from $(D, X \cup P^i)$.

The Repairing Algorithm

The following observations imply the termination of our algorithm:

- There are finitely many background points in (D, X), and there are also finitely many critical configurations in (D, X).
- Only background points (from $D \setminus P^i$) are inserted into P^i, and no point is removed from P^i.
- Every iteration i of our algorithm inserts at least one point into P^i, and each point insertion eliminates at least one critical configuration from $(D, X \cup P^i)$.
- Since background points are never removed from P, a critical configuration that is eliminated at iteration i cannot show up again at iteration $j > i$.

The Repairing Algorithm

The following observations imply the termination of our algorithm:

- There are finitely many background points in (D, X), and there are also finitely many critical configurations in (D, X).
- Only background points (from $D \setminus P^i$) are inserted into P^i, and no point is removed from P^i.
- Every iteration i of our algorithm inserts at least one point into P^i, and each point insertion eliminates at least one critical configuration from $(D, X \cup P^i)$.
- Since background points are never removed from P, a critical configuration that is eliminated at iteration i cannot show up again at iteration $j > i$.

The correctness of our algorithm follows from the fact that it terminates, as $(D, X \cup P^i) = (D, X \cup P)$ contains no critical configurations after the last iteration i of the algorithm.

The Repairing Algorithm

Our algorithm is linear in the number $|D|$ of points of (D, X) as:

The Repairing Algorithm

Our algorithm is linear in the number $|D|$ of points of (D, X) as:

- The search for critical configurations is done in $\mathcal{O}(|D|)$ in the first iteration, and there is no ''search'' in the next iterations.

The Repairing Algorithm

Our algorithm is linear in the number $|D|$ of points of (D, X) as:

- The search for critical configurations is done in $\mathcal{O}(|D|)$ in the first iteration, and there is no ‘‘search’’ in the next iterations.

- There can be at most $n - 1$ iterations, where $n \leq |D|$ is the number of background points of (D, X), as each iteration i inserts a distinct background point from $D \setminus P^i$ into P^i.

The Repairing Algorithm

Our algorithm is linear in the number $|D|$ of points of (D, X) as:

- The search for critical configurations is done in $\mathcal{O}(|D|)$ in the first iteration, and there is no ‘‘search’’ in the next iterations.
- There can be at most $n - 1$ iterations, where $n \leq |D|$ is the number of background points of (D, X), as each iteration i inserts a distinct background point from $D \setminus P^i$ into P^i.

What about the size of P?

The Repairing Algorithm

Our algorithm is linear in the number $|D|$ of points of (D, X) as:

- The search for critical configurations is done in $\mathcal{O}(|D|)$ in the first iteration, and there is no ''search'' in the next iterations.
- There can be at most $n - 1$ iterations, where $n \leq |D|$ is the number of background points of (D, X), as each iteration i inserts a distinct background point from $D \setminus P^i$ into P^i.

What about the size of P?

Note that the size of P is bounded above by the number of **all** critical configurations eliminated by the algorithm, as each point inserted into P eliminates at least one critical configuration.

The Repairing Algorithm

Regard the size of P, we have the following result:

The Repairing Algorithm

Regard the size of P, we have the following result:

Theorem. *Let (D, X) be a 3D binary digital image. If there are exactly m instances of critical configurations in (D, X), then the* **expected value** *$E[t]$ of the number t of new critical configurations created by the repairing algorithm described before on input (D, X) is less than $m/2$.*

The Repairing Algorithm

Regard the size of P, we have the following result:

Theorem. *Let (D, X) be a 3D binary digital image. If there are exactly m instances of critical configurations in (D, X), then the* **expected value** *$E[t]$ of the number t of new critical configurations created by the repairing algorithm described before on input (D, X) is less than $m/2$.*

Since the size of P is bounded above by the number of critical configurations in (D, X), the above theorem tells us that the expected value $E[|P|]$ of the size of P is $E[|P|] < 3m/2$.

The Repairing Algorithm

Regard the size of P, we have the following result:

Theorem. *Let* (D, X) *be a 3D binary digital image. If there are exactly* m *instances of critical configurations in* (D, X)*, then the* **expected value** $E[t]$ *of the number* t *of new critical configurations created by the repairing algorithm described before on input* (D, X) *is less than* $m/2$*.*

Since the size of P is bounded above by the number of critical configurations in (D, X), the above theorem tells us that the expected value $E[|P|]$ of the size of P is $E[|P|] < 3m/2$.

So, in principle, if the number of critical configurations is small compared to the number of points of the image, our algorithm is expected to produce a well-composed image that is similar to the input ill-composed one.

Experimental Results

We applied our algorithm to six distinct 3D binary digital images.

Experimental Results

We applied our algorithm to six distinct 3D binary digital images.

Each of the six images has $129^3 = 2{,}146{,}689$ points and was obtained from segmenting and resampling grey-scale MR images.

Experimental Results

We applied our algorithm to six distinct 3D binary digital images.

Each of the six images has $129^3 = 2{,}146{,}689$ points and was obtained from segmenting and resampling grey-scale MR images.

Three of the binary images were obtained from three distinct segmentations (white matter, grey matter, and CSF) of a normal brain MR image produced by an on-line 3D MR image simulator (*www.bic.mni.mgill.ca/brainweb*).

Experimental Results

We applied our algorithm to six distinct 3D binary digital images.

Each of the six images has $129^3 = 2{,}146{,}689$ points and was obtained from segmenting and resampling grey-scale MR images.

Three of the binary images were obtained from three distinct segmentations (white matter, grey matter, and CSF) of a normal brain MR image produced by an on-line 3D MR image simulator (*www.bic.mni.mgill.ca/brainweb*).

Two other images are real lung images corresponding to the inspiration and expiration stages of lung motion.

Experimental Results

We applied our algorithm to six distinct 3D binary digital images.

Each of the six images has $129^3 = 2{,}146{,}689$ points and was obtained from segmenting and resampling grey-scale MR images.

Three of the binary images were obtained from three distinct segmentations (white matter, grey matter, and CSF) of a normal brain MR image produced by an on-line 3D MR image simulator (*www.bic.mni.mgill.ca/brainweb*).

Two other images are real lung images corresponding to the inspiration and expiration stages of lung motion.

The sixth image is a male thorax image from the dataset of the Visible Human Project (*www.nlm.nih.gov/research/visible*).

Experimental Results

We ran our algorithm 10 times on each of the six images.

Experimental Results

We ran our algorithm 10 times on each of the six images.

For each execution, we kept track of the size of P and of the number of critical configurations created by the algorithm. Then, we computed the average size of P and the average number of critical configurations created by the algorithm for the 10 executions of our algorithm on each image.

Experimental Results

We ran our algorithm 10 times on each of the six images.

For each execution, we kept track of the size of P and of the number of critical configurations created by the algorithm. Then, we computed the average size of P and the average number of critical configurations created by the algorithm for the 10 executions of our algorithm on each image.

For each of the six images, the average size of P was at most 0.32% of the size of the image.

Experimental Results

We ran our algorithm 10 times on each of the six images.

For each execution, we kept track of the size of P and of the number of critical configurations created by the algorithm. Then, we computed the average size of P and the average number of critical configurations created by the algorithm for the 10 executions of our algorithm on each image.

For each of the six images, the average size of P was at most 0.32% of the size of the image.

The average number of new critical configurations is no larger than $0.2 \cdot m$, where m is the number of critical configuration in the input image.

Experimental Results

For each of the six images, the average size of P is smaller than m, which means that the size of P and the size of the smallest possible P are close.

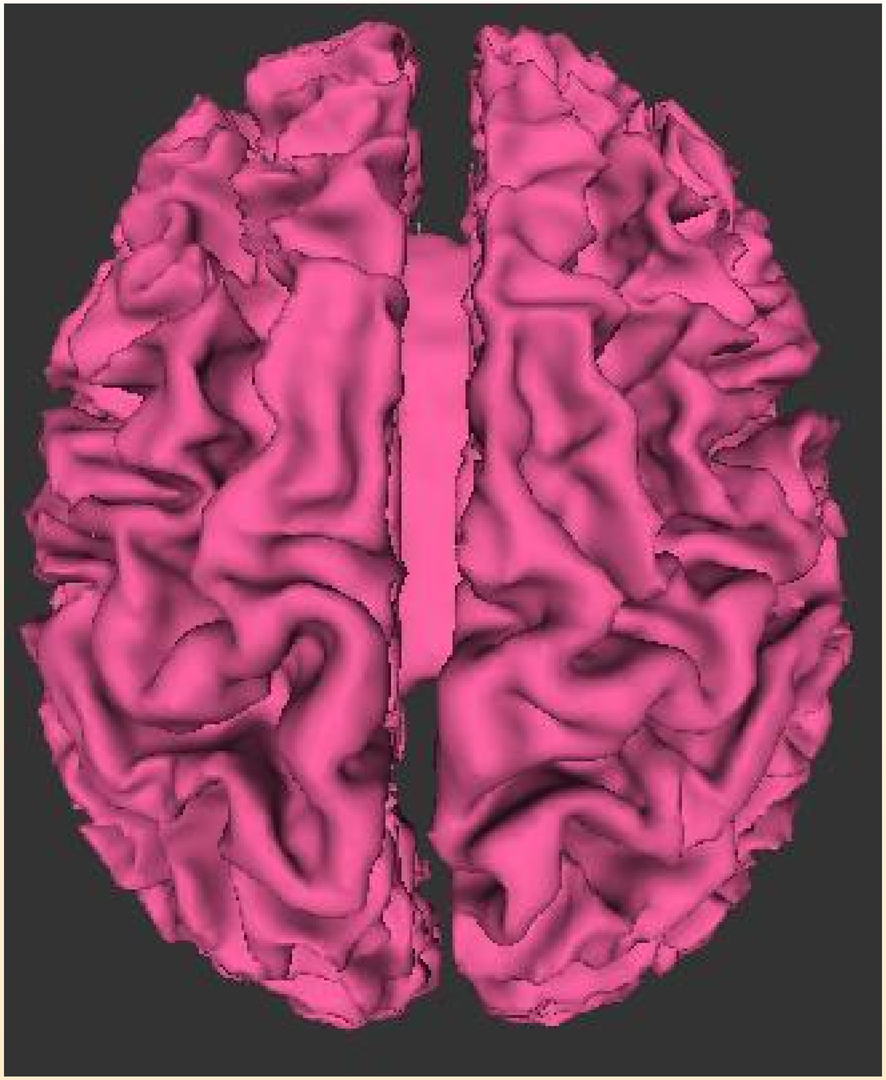
Experimental Results

For each of the six images, the average size of P is smaller than m, which means that the size of P and the size of the smallest possible P are close.

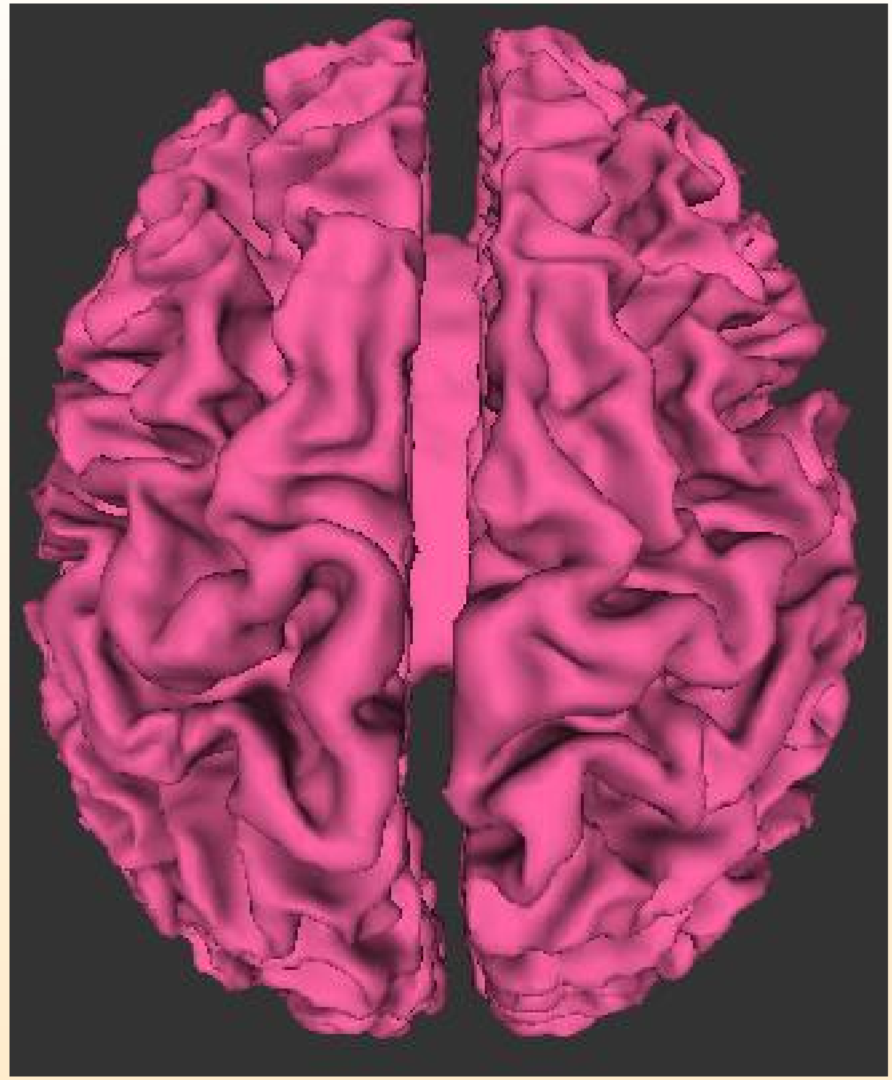
Code available at `http://www.seas.upenn.edu/~marcelos/wellcomp.html`.

Image	# C. Conf. 1)	# C. Conf. 2)	Avg. $|\mathbf{P}|$	Avg. # New C. C.
Brain (White Matter)	2538	418	1833.5	249.9
Brain (Grey Matter)	9688	1316	6834.2	2115.9
Brain (CSF)	8574	676	5303.3	787.1
Lung (Inspiration)	1908	128	1213.7	288.4
Lung (Expiration)	3284	237	2068.7	483.2
Thorax	1962	84	1123.7	143.8

Experimental Results

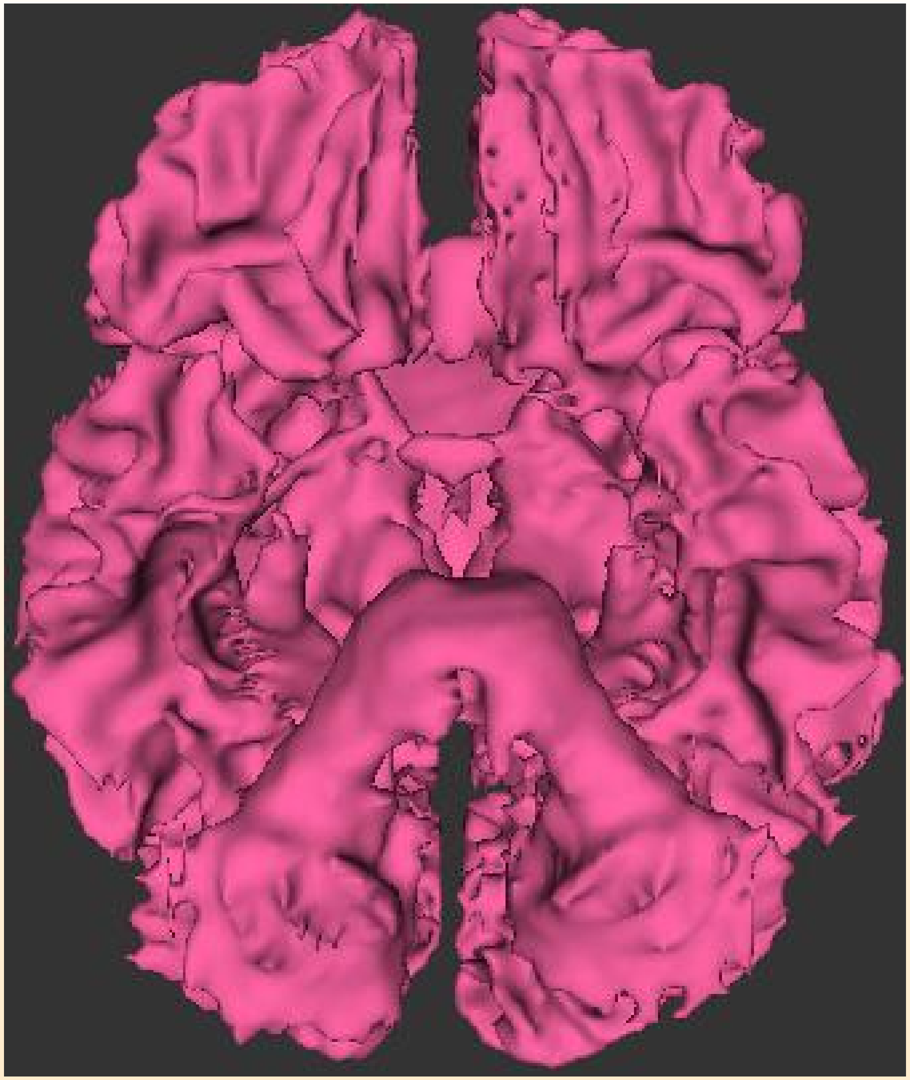


From ill-composed image

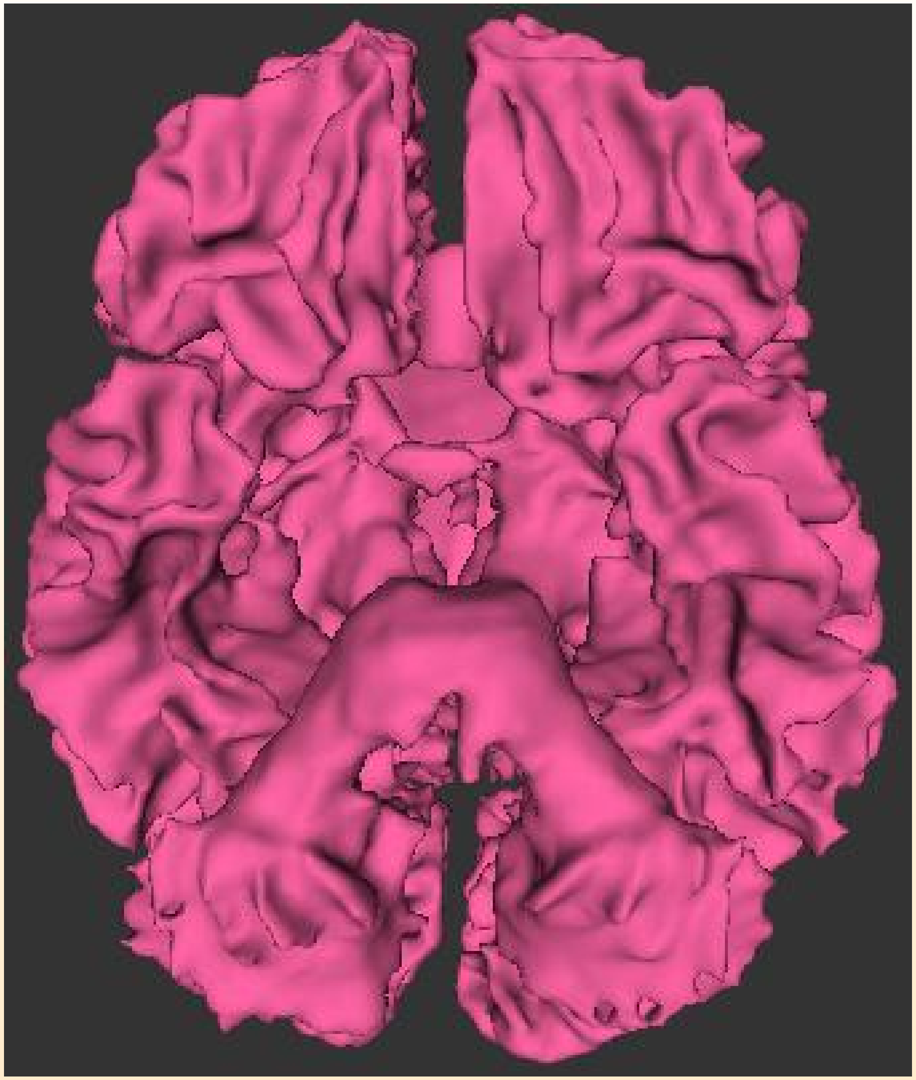


From well-composed image

Experimental Results



From ill-composed image



From well-composed image

Conclusions

We provided a simple randomized algorithm for repairing ill-composed images in linear time in the number of points of the image.

Conclusions

We provided a simple randomized algorithm for repairing ill-composed images in linear time in the number of points of the image.

We provided theoretical evidence that our algorithm is very likely to produce a well-composed image similar to the corresponding ill-composed one.

Conclusions

We provided a simple randomized algorithm for repairing ill-composed images in linear time in the number of points of the image.

We provided theoretical evidence that our algorithm is very likely to produce a well-composed image similar to the corresponding ill-composed one.

We also provided experimental evidence of the effectiveness of our repairing algorithm when faced with real data from medical applications.

Conclusions

We provided a simple randomized algorithm for repairing ill-composed images in linear time in the number of points of the image.

We provided theoretical evidence that our algorithm is very likely to produce a well-composed image similar to the corresponding ill-composed one.

We also provided experimental evidence of the effectiveness of our repairing algorithm when faced with real data from medical applications.

For future work, we intend to (1) investigate the existence of a linear time algorithm for finding the smallest set P; (2) include a "look ahead" mechanism in our algorithm to reduce the size of P; and (3) extend our repairing algorithm to handle multicolor images.